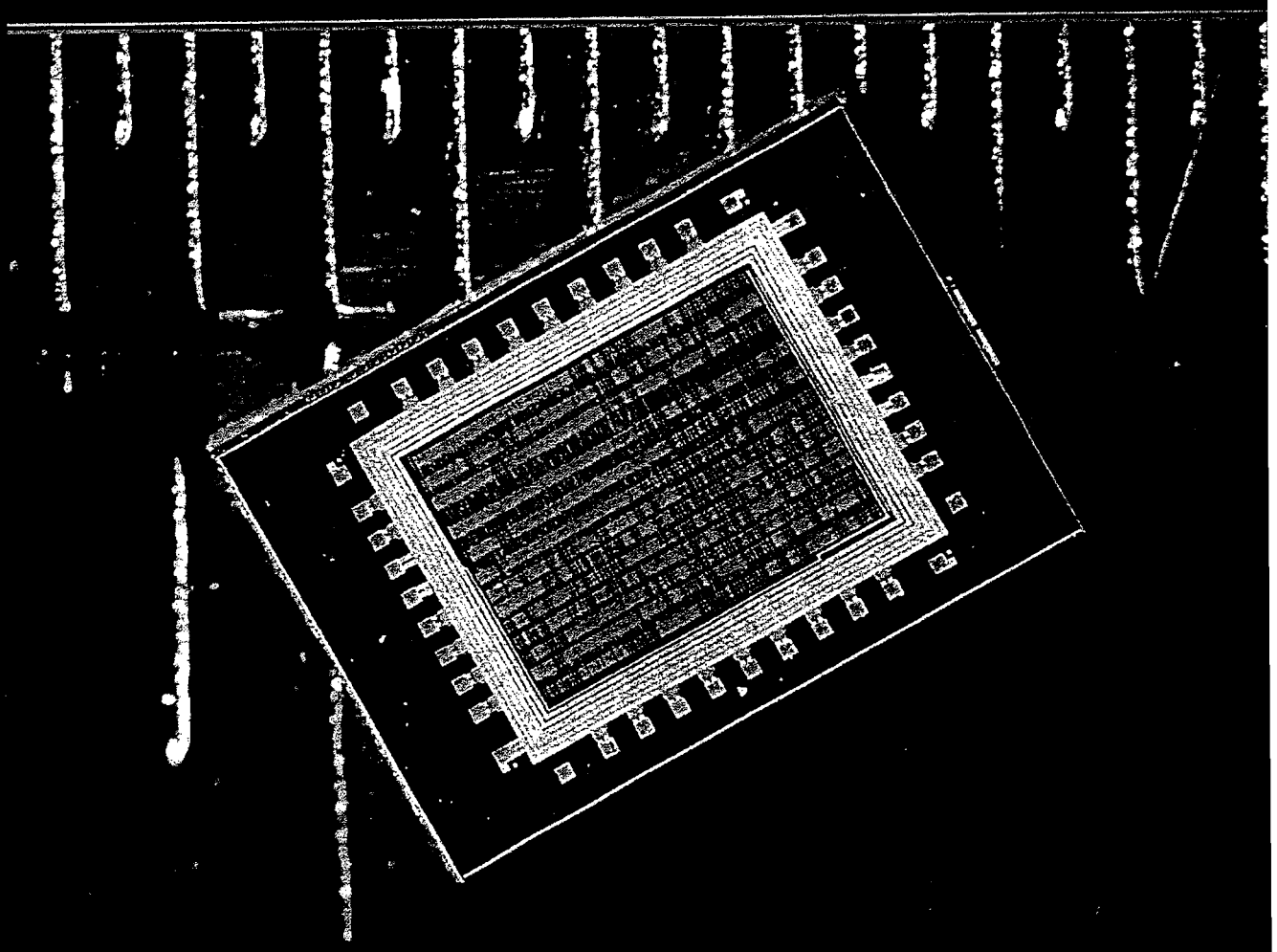
EXHIBIT 33

Application-Specific Integrated Circuits

Michael John Sebastian Smith



Application-Specific Integrated Circuits

Michael John Sebastian Smith

ADDISON-WESLEY

An imprint of Addison Wesley Longman, Inc.

Reading, Massachusetts • Harlow, England • Menlo Park, California • Berkeley, California
Don Mills, Ontario • Sydney • Bonn • Amsterdam • Tokyo • Mexico City

This book is in the Addison-Wesley VLSI Systems Series
Lynn Conway and Charles Seitz, *Consulting Editors*

Sponsoring Editor	Peter Gordon
Associate Editor	Helen Goldstein
Senior Production Supervisor	Juliet Silveri
Copyeditor/Proofreader	Cynthia Benn
Cover Design Supervisor	Simone Payment
Marketing Manager	Tracy Russ
Manufacturing Manager	Roy Logan

Library of Congress Cataloging-in-Publication Data

Smith, Michael J. S. (Michael John Sebastian)
Application-specific integrated circuits / Michael J.S. Smith.
p. cm.
Includes bibliographical references and index.
ISBN 0-201-50022-1
1. Application-specific integrated circuits. I. Title.
TK7874.6.S63 1997
621.39'5--dc20 93-32538
CIP

Access the latest information about Addison-Wesley books from our World Wide Web page:
`http://www.awl.com/cseng`

Text printed on recycled and acid-free paper.

ISBN 0201500221

7 8 9 101112 CRW 02 01 00 99

7th Printing November 1999

16 FLOORPLANNING AND PLACEMENT

The input to the floorplanning step is the output of system partitioning and design entry—a netlist. Floorplanning precedes placement, but we shall cover them together. The output of the placement step is a set of directions for the routing tools.

At the start of floorplanning we have a netlist describing circuit blocks, the logic cells within the blocks, and their connections. For example, Figure 16.1 shows the Viterbi decoder example as a collection of standard cells with no room set aside yet for routing. We can think of the standard cells as a hod of bricks to be made into a wall. What we have to do now is set aside spaces (we call these spaces the **channels**) for interconnect, the mortar, and arrange the cells. Figure 16.2 shows a finished wall—after floorplanning and placement steps are complete. We still have not completed any routing at this point—that comes later—all we have done is placed the logic cells in a fashion that we hope will minimize the total interconnect length, for example.

16.1 Floorplanning

Figure 16.3 shows that both interconnect delay and gate delay decrease as we scale down feature sizes—but at different rates. This is because interconnect capacitance tends to a limit of about $2\,\mathrm{pFcm}^{-1}$ for a minimum-width wire while gate delay continues to decrease (see Section 17.4, "Circuit Extraction and DRC"). Floorplanning allows us to predict this interconnect delay by estimating interconnect length.

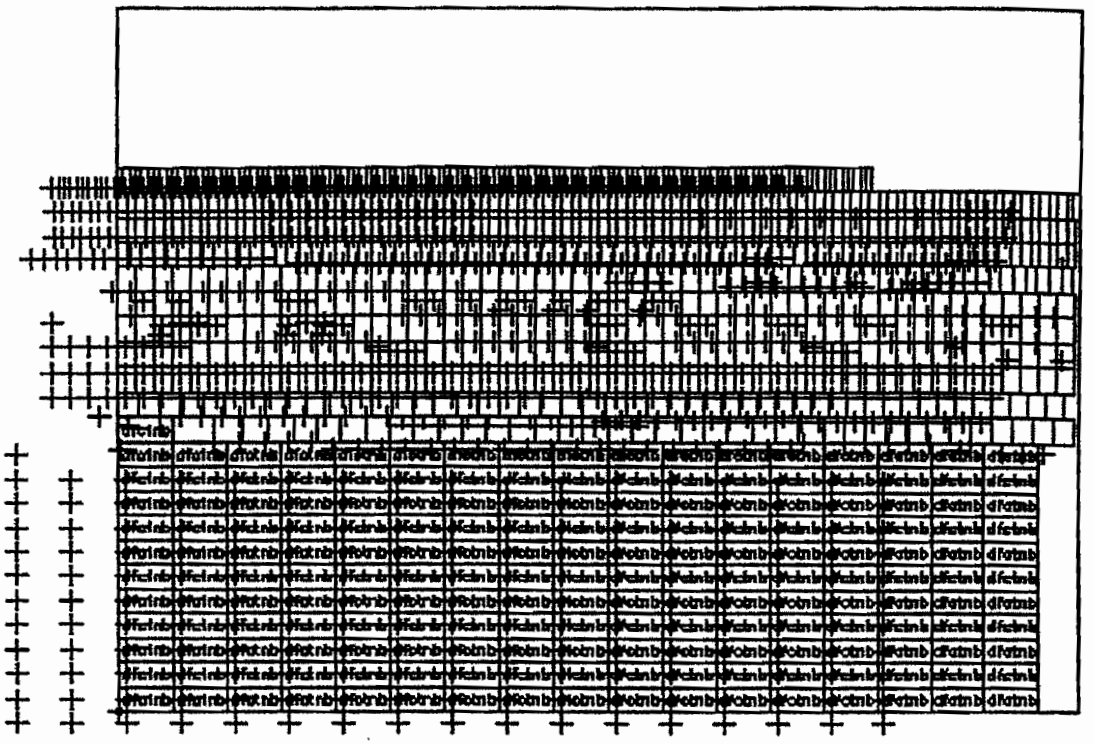


FIGURE 16.1 The starting point for the floorplanning and placement steps for the Viterbi decoder (containing only standard cells). This is the initial display of the floorplanning and placement tool. The small boxes that look like bricks are the outlines of the standard cells. The largest standard cells, at the bottom of the display (labeled dfctnb) are 188 D flip-flops. The '+' symbols represent the drawing origins of the standard cells—for the D flip-flops they are shifted to the left and below the logic cell bottom left-hand corner. The large box surrounding all the logic cells represents the estimated chip size. (This is a screen shot from Cadence Cell Ensemble.)

16.1.1 Floorplanning Goals and Objectives

The input to a floorplanning tool is a hierarchical netlist that describes the interconnection of the **blocks** (RAM, ROM, ALU, cache controller, and so on); the logic cells (NAND, NOR, D flip-flop, and so on) within the blocks; and the logic cell connectors (the terms *terminals, pins*, or *ports* mean the same thing as *connectors*). The netlist is a logical description of the ASIC; the floorplan is a physical description of an ASIC. Floorplanning is thus a mapping between the logical description (the netlist) and the physical description (the floorplan).

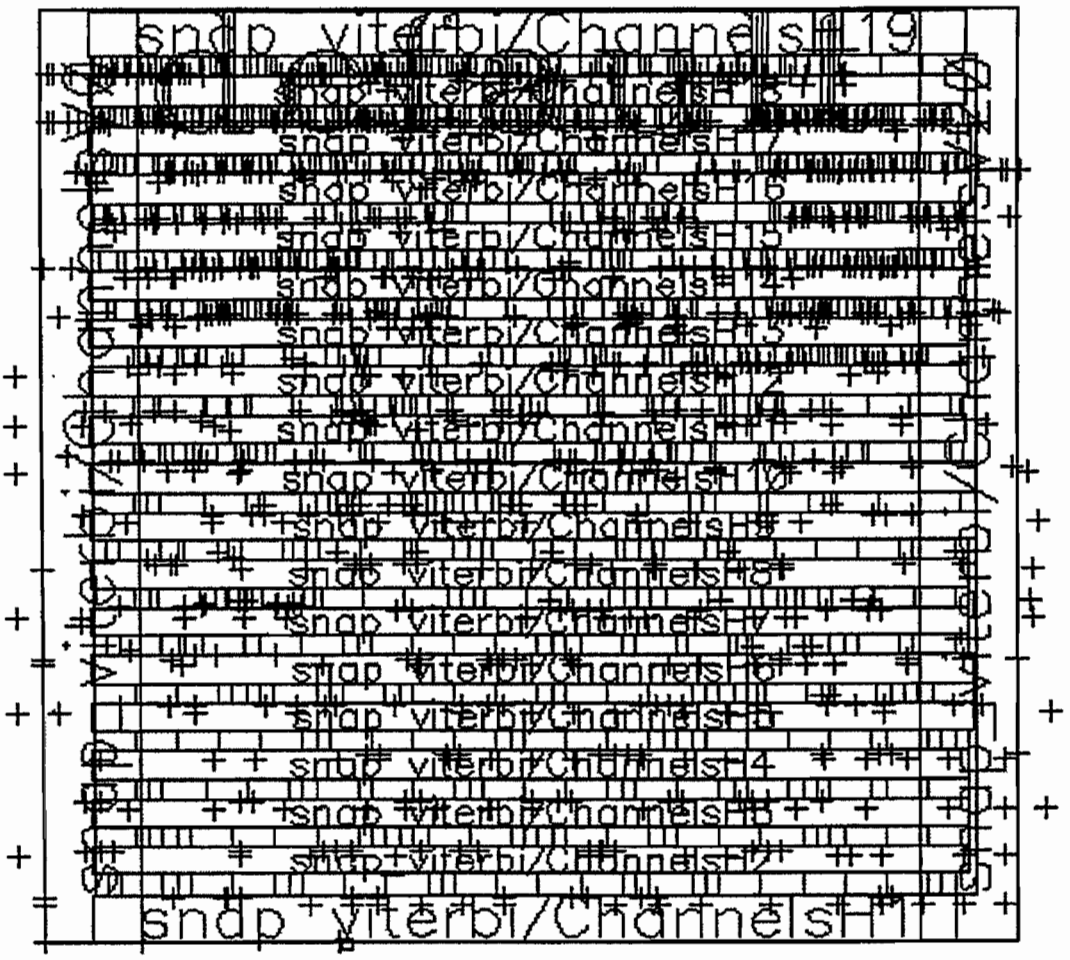


FIGURE 16.2 The Viterbi Decoder (from Figure 16.1) after floorplanning and placement. There are 18 rows of standard cells separated by 17 horizontal channels (labeled 2–18). The channels are routed as numbered. In this example, the I/O pads are omitted to show the cell placement more clearly. Figure 17.1 shows the same placement without the channel labels. (A screen shot from Cadence Cell Ensemble.)

The goals of floorplanning are to:

- arrange the blocks on a chip,
- decide the location of the I/O pads,
- decide the location and number of the power pads,

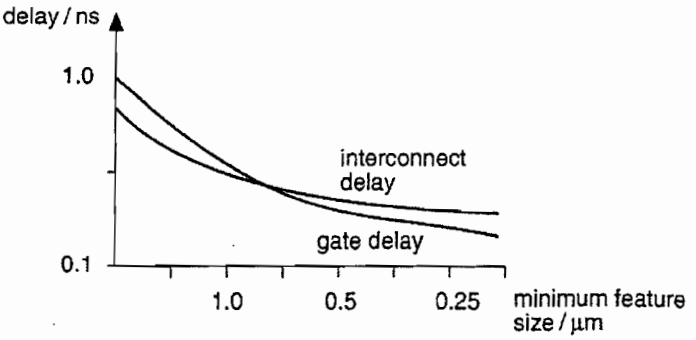


FIGURE 16.3 Interconnect and gate delays. As feature sizes decrease, both average interconnect delay and average gate delay decrease—but at different rates. This is because interconnect capacitance tends to a limit that is independent of scaling. Interconnect delay now dominates gate delay.

- decide the type of power distribution, and
- decide the location and type of clock distribution.

The objectives of floorplanning are to minimize the chip area and minimize delay. Measuring area is straightforward, but measuring delay is more difficult and we shall explore this next.

16.1.2 Measurement of Delay in Floorplanning

Throughout the ASIC design process we need to predict the performance of the final layout. In floorplanning we wish to predict the interconnect delay before we complete any routing. Imagine trying to predict how long it takes to get from Russia to China without knowing where in Russia we are or where our destination is in China. Actually it is worse, because in floorplanning we may move Russia or China.

To predict delay we need to know the **parasitics** associated with interconnect: the **interconnect capacitance** (**wiring capacitance** or **routing capacitance**) as well as the interconnect resistance. At the floorplanning stage we know only the **fanout** (**FO**) of a net (the number of gates driven by a net) and the size of the block that the net belongs to. We cannot predict the resistance of the various pieces of the interconnect path since we do not yet know the shape of the interconnect for a net. However, we can estimate the total length of the interconnect and thus estimate the total capacitance. We estimate interconnect length by collecting statistics from previously routed chips and analyzing the results. From these statistics we create tables that predict the interconnect capacitance as a function of net fanout and block size. A floorplanning tool can then use these **predicted-capacitance tables** (also known as **interconnect-load tables** or **wire-load tables**). Figure 16.4 shows how we derive and use wire-load tables and illustrates the following facts:

- Typically between 60 and 70 percent of nets have a FO = 1.
- The distribution for a FO = 1 has a very long tail, stretching to interconnects that run from corner to corner of the chip.

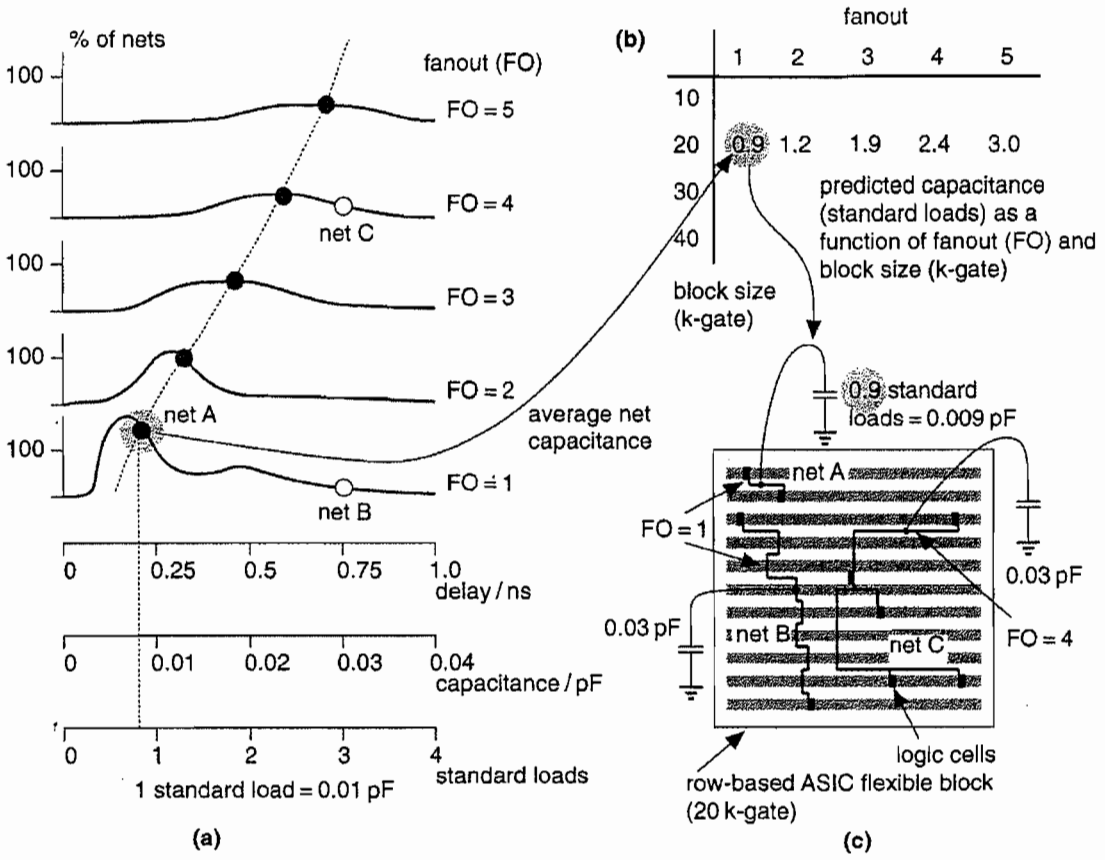


FIGURE 16.4 Predicted capacitance. (a) Interconnect lengths as a function of fanout (FO) and circuit-block size. (b) Wire-load table. There is only one capacitance value for each fanout (typically the average value). (c) The wire-load table predicts the capacitance and delay of a net (with a considerable error). Net A and net B both have a fanout of 1, both have the same predicted net delay, but net B in fact has a much greater delay than net A in the actual layout (of course we shall not know what the actual layout is until much later in the design process).

- The distribution for a FO = 1 often has two peaks, corresponding to a distribution for close neighbors in subgroups within a block, superimposed on a distribution corresponding to routing between subgroups.
- We often see a twin-peaked distribution at the chip level also, corresponding to separate distributions for **interblock routing** (inside blocks) and **intrablock routing** (between blocks).

- The distributions for FO > 1 are more symmetrical and flatter than for FO = 1.
- The wire-load tables can only contain one number, for example the average net capacitance, for any one distribution. Many tools take a worst-case approach and use the 80- or 90-percentile point instead of the average. Thus a tool may use a predicted capacitance for which we know 90 percent of the nets will have less than the estimated capacitance.
- We need to repeat the statistical analysis for blocks with different sizes. For example, a net with a FO = 1 in a 25 k-gate block will have a different (larger) average length than if the net were in a 5 k-gate block.
- The statistics depend on the shape (aspect ratio) of the block (usually the statistics are only calculated for square blocks).
- The statistics will also depend on the type of netlist. For example, the distributions will be different for a netlist generated by setting a constraint for minimum logic delay during synthesis—which tends to generate large numbers of two-input NAND gates—than for netlists generated using minimum-area constraints.

There are no standards for the wire-load tables themselves, but there are some standards for their use and for presenting the extracted loads (see Section 16.4). Wire-load tables often present loads in terms of a **standard load** that is usually the input capacitance of a two-input NAND gate with a 1X (default) drive strength.

TABLE 16.1 A wire-load table showing average interconnect lengths (mm).[1]

		Fanout		
Array (available gates)	Chip size (mm)	1	2	4
3 k	3.45	0.56	0.85	1.46
11 k	5.11	0.84	1.34	2.25
105 k	12.50	1.75	2.70	4.92

[1] Interconnect lengths are derived from interconnect capacitance data. Interconnect capacitance is 2 $pFcm^{-1}$.

Table 16.1 shows the estimated metal interconnect lengths, as a function of die size and fanout, for a series of three-level metal gate arrays. In this case the interconnect capacitance is about 2 $pFcm^{-1}$, a typical figure.

Figure 16.5 shows that, because we do not decrease chip size as we scale down feature size, the worst-case interconnect delay increases. One way to measure the worst-case delay uses an interconnect that completely crosses the chip, a **coast-to-coast interconnect**. In certain cases the worst-case delay of a 0.25 μm process may be worse than a 0.35 μm process, for example.

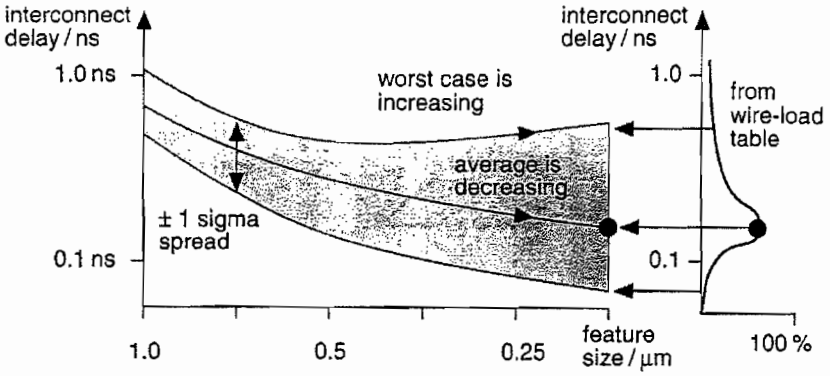


FIGURE 16.5 Worst-case interconnect delay. As we scale circuits, but avoid scaling the chip size, the worst-case interconnect delay increases.

16.1.3 Floorplanning Tools

Figure 16.6(a) shows an initial **random floorplan** generated by a floorplanning tool. Two of the blocks, A and C in this example, are standard-cell areas (the chip shown in Figure 16.1 is one large standard-cell area). These are **flexible blocks** (or **variable blocks**) because, although their total area is fixed, their shape (aspect ratio) and connector locations may be adjusted during the placement step. The dimensions and connector locations of the other **fixed blocks** (perhaps RAM, ROM, compiled cells, or megacells) can only be modified when they are created. We may force logic cells to be in selected flexible blocks by **seeding**. We choose **seed cells** by name. For example, `ram_control*` would select all logic cells whose names started with `ram_control` to be placed in one flexible block. The special symbol, usually '*', is a **wildcard symbol**. Seeding may be hard or soft. A **hard seed** is fixed and not allowed to move during the remaining floorplanning and placement steps. A **soft seed** is an initial suggestion only and can be altered if necessary by the floorplanner. We may also use **seed connectors** within flexible blocks—forcing certain nets to appear in a specified order, or location at the boundary of a flexible block.

The floorplanner can complete an estimated placement to determine the positions of connectors at the boundaries of the flexible blocks. Figure 16.6(b) illustrates a **rat's nest** display of the connections between blocks. Connections are shown as **bundles** between the centers of blocks or as **flight lines** between connectors. Figure 16.6(c) and (d) show how we can move the blocks in a floorplanning tool to minimize routing **congestion**.

We need to control the **aspect ratio** of our floorplan because we have to fit our chip into the **die cavity** (a fixed-size hole, usually square) inside a package. Figure 16.7(a)–(c) show how we can rearrange our chip to achieve a square aspect ratio. Figure 16.7(c) also shows a **congestion map**, another form of **routability** display. There is no standard measure of routability. Generally the **interconnect channels**, (or wiring channels—I shall call them channels from now on) have a cer-

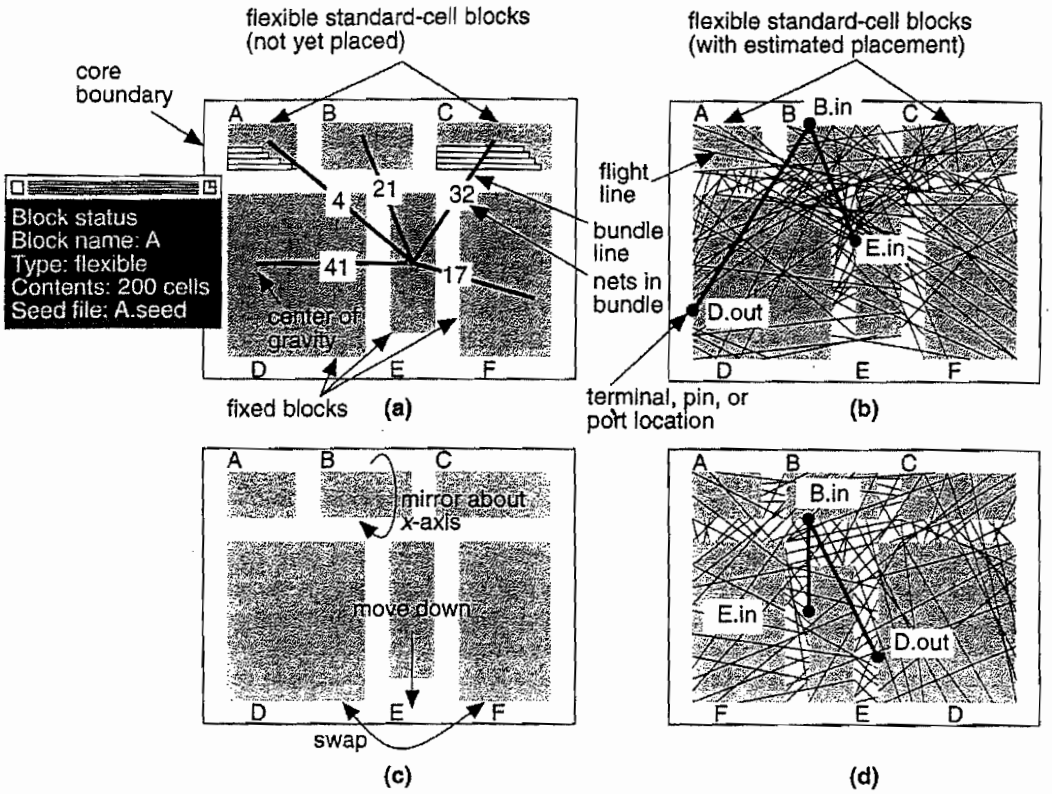


FIGURE 16.6 Floorplanning a cell-based ASIC. (a) Initial floorplan generated by the floorplanning tool. Two of the blocks are flexible (A and C) and contain rows of standard cells (unplaced). A pop-up window shows the status of block A. (b) An estimated placement for flexible blocks A and C. The connector positions are known and a rat's nest display shows the heavy congestion below block B. (c) Moving blocks to improve the floorplan. (d) The updated display shows the reduced congestion after the changes.

tain **channel capacity**; that is, they can handle only a fixed number of interconnects. One measure of congestion is the difference between the number of interconnects that we actually need, called the **channel density**, and the channel capacity. Another measure, shown in Figure 16.7(c), uses the ratio of channel density to the channel capacity. With practice, we can create a good initial placement by floorplanning and a pictorial display. This is one area where the human ability to recognize patterns and spatial relations is currently superior to a computer program's ability.

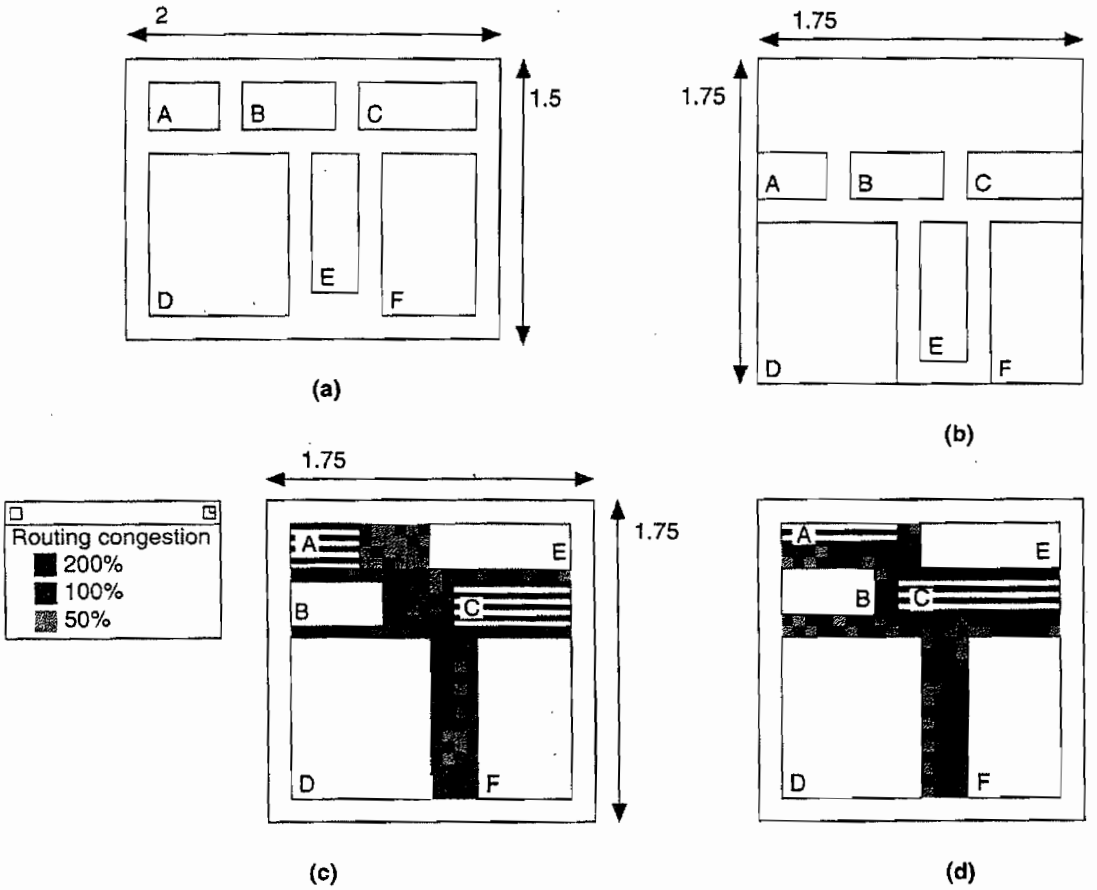


FIGURE 16.7 Congestion analysis. (a) The initial floorplan with a 2:1.5 die aspect ratio. (b) Altering the floorplan to give a 1:1 chip aspect ratio. (c) A trial floorplan with a congestion map. Blocks A and C have been placed so that we know the terminal positions in the channels. Shading indicates the ratio of channel density to the channel capacity. Dark areas show regions that cannot be routed because the channel congestion exceeds the estimated capacity. (d) Resizing flexible blocks A and C alleviates congestion.

16.1.4 Channel Definition

During the floorplanning step we assign the areas between blocks that are to be used for interconnect. This process is known as **channel definition** or **channel allocation**. Figure 16.8 shows a T-shaped junction between two rectangular channels

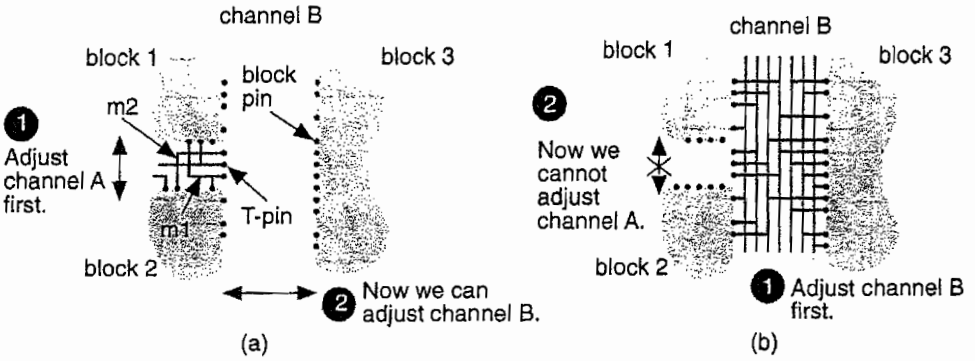


FIGURE 16.8 Routing a T-junction between two channels in two-level metal. The dots represent logic cell pins. (a) Routing channel A (the stem of the T) first allows us to adjust the width of channel B. (b) If we route channel B first (the top of the T), this fixes the width of channel A. We have to route the stem of a T-junction before we route the top.

and illustrates why we must route the stem (vertical) of the T before the bar. The general problem of choosing the order of rectangular channels to route is **channel ordering**.

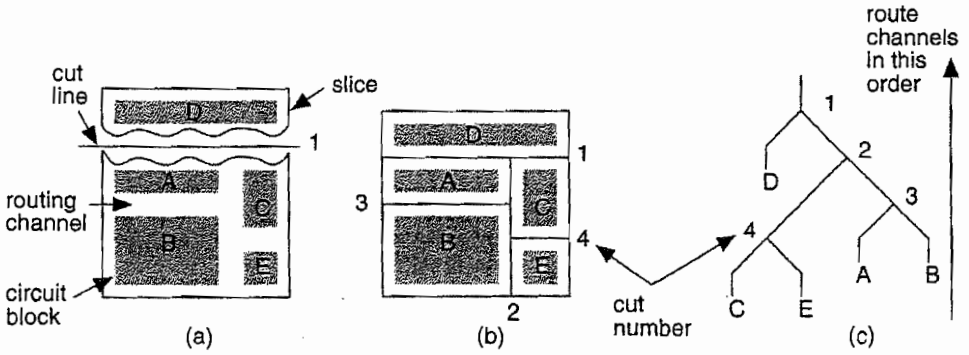


FIGURE 16.9 Defining the channel routing order for a slicing floorplan using a slicing tree. (a) Make a cut all the way across the chip between circuit blocks. Continue slicing until each piece contains just one circuit block. Each cut divides a piece into two without cutting through a circuit block. (b) A sequence of cuts: 1, 2, 3, and 4 that successively slices the chip until only circuit blocks are left. (c) The slicing tree corresponding to the sequence of cuts gives the order in which to route the channels: 4, 3, 2, and finally 1.

Figure 16.9 shows a floorplan of a chip containing several blocks. Suppose we cut along the block boundaries slicing the chip into two pieces (Figure 16.9a). Then suppose we can slice each of these pieces into two. If we can continue in this fashion until all the blocks are separated, then we have a **slicing floorplan** (Figure 16.9b). Figure 16.9(c) shows how the sequence we use to slice the chip defines a hierarchy of the blocks. Reversing the slicing order ensures that we route the stems of all the channel T-junctions first.

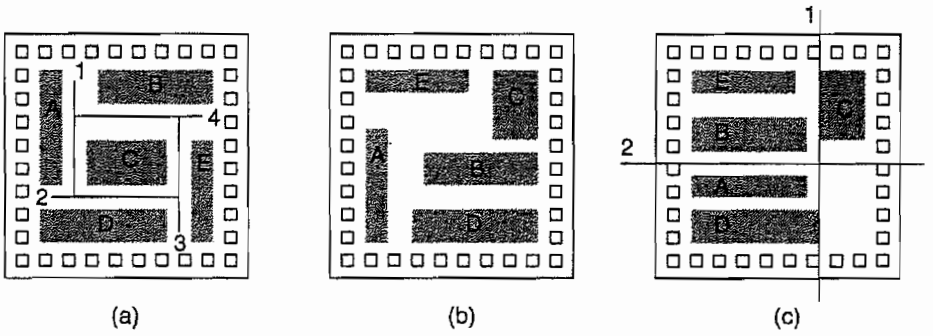


FIGURE 16.10 Cyclic constraints. (a) A nonslicing floorplan with a cyclic constraint that prevents channel routing. (b) In this case it is difficult to find a slicing floorplan without increasing the chip area. (c) This floorplan may be sliced (with initial cuts 1 or 2) and has no cyclic constraints, but it is inefficient in area use and will be very difficult to route.

Figure 16.10 shows a floorplan that is not a slicing structure. We cannot cut the chip all the way across with a knife without chopping a circuit block in two. This means we cannot route any of the channels in this floorplan without routing all of the other channels first. We say there is a **cyclic constraint** in this floorplan. There are two solutions to this problem. One solution is to move the blocks until we obtain a slicing floorplan. The other solution is to allow the use of L-shaped, rather than rectangular, channels (or areas with fixed connectors on all sides—a **switch box**). We need an area-based router rather than a channel router to route L-shaped regions or switch boxes (see Section 17.2.6, "Area-Routing Algorithms").

Figure 16.11(a) displays the floorplan of the ASIC shown in Figure 16.7. We can remove the cyclic constraint by moving the blocks again, but this increases the chip size. Figure 16.11(b) shows an alternative solution. We **merge** the flexible standard cell areas A and C. We can do this by **selective flattening** of the netlist. Sometimes flattening can reduce the routing area because routing between blocks is usually less efficient than routing inside the row-based blocks. Figure 16.11(b) shows the channel definition and **routing order** for our chip.

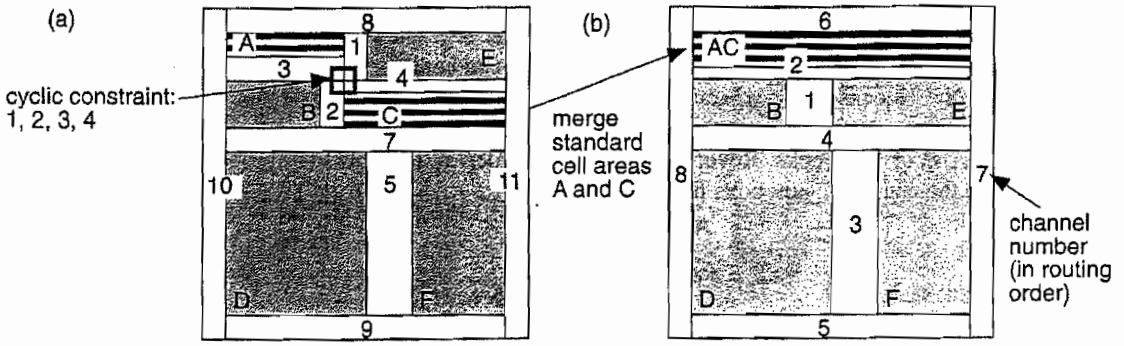


FIGURE 16.11 Channel definition and ordering. (a) We can eliminate the cyclic constraint by merging the blocks A and C. (b) A slicing structure.

16.1.5 I/O and Power Planning

Every chip communicates with the outside world. Signals flow onto and off the chip and we need to supply power. We need to consider the I/O and power constraints early in the floorplanning process. A silicon chip or **die** (plural die, dies, or dice) is mounted on a **chip carrier** inside a chip **package**. Connections are made by **bonding** the chip **pads** to fingers on a metal **lead frame** that is part of the package. The metal lead-frame fingers connect to the **package pins**. A die consists of a logic **core** inside a **pad ring**. Figure 16.12(a) shows a **pad-limited die** and Figure 16.12(b) shows a **core-limited die**. On a pad-limited die we use tall, thin **pad-limited pads**, which maximize the number of pads we can fit around the outside of the chip. On a core-limited die we use short, wide **core-limited pads**. Figure 16.12(c) shows how we can use both types of pad to change the aspect ratio of a die to be different from that of the core.

Special **power pads** are used for the positive supply, or VDD, **power buses** (or **power rails**) and the ground or negative supply, VSS or GND. Usually one set of VDD/VSS pads supplies one **power ring** that runs around the pad ring and supplies power to the I/O pads only. Another set of VDD/VSS pads connects to a second power ring that supplies the logic core. We sometimes call the I/O power **dirty power** since it has to supply large transient currents to the output transistors. We keep dirty power separate to avoid injecting noise into the internal-logic power (the **clean power**). I/O pads also contain special circuits to protect against **electrostatic discharge** (**ESD**). These circuits can withstand very short high-voltage (several kilovolt) pulses that can be generated during human or machine handling.

Depending on the type of package and how the foundry attaches the silicon die to the **chip cavity** in the chip carrier, there may be an electrical connection between the chip carrier and the die substrate. Usually the die is cemented in the chip cavity with a conductive epoxy, making an electrical connection between substrate and the

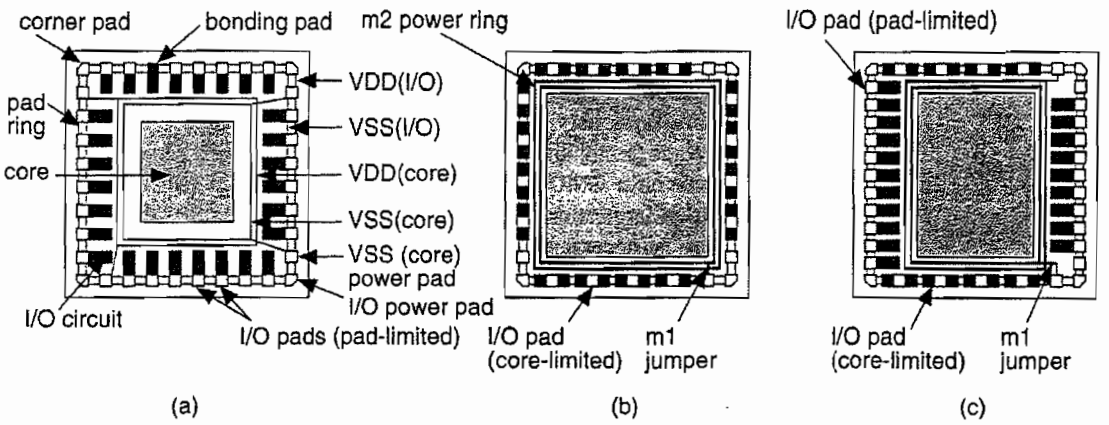


FIGURE 16.12 Pad-limited and core-limited die. (a) A pad-limited die. The number of pads determines the die size. (b) A core-limited die: The core logic determines the die size. (c) Using both pad-limited pads and core-limited pads for a square die.

package cavity in the chip carrier. If we make an electrical connection between the substrate and a chip pad, or to a package pin, it must be to VDD (*n*-type substrate) or VSS (*p*-type substrate). This **substrate connection** (for the whole chip) employs a **down bond** (or drop bond) to the carrier. We have several options:

- We can dedicate one (or more) chip pad(s) to down bond to the chip carrier.
- We can make a connection from a chip pad to the lead frame and down bond from the chip pad to the chip carrier.
- We can make a connection from a chip pad to the lead frame and down bond from the lead frame.
- We can down bond from the lead frame without using a chip pad.
- We can leave the substrate and/or chip carrier unconnected.

Depending on the package design, the type and positioning of down bonds may be fixed. This means we need to fix the position of the chip pad for down bonding using a **pad seed**.

A **double bond** connects two pads to one chip-carrier finger and one package pin. We can do this to save package pins or reduce the series inductance of bond wires (typically a few nanohenries) by parallel connection of the pads. A **multiple-signal pad** or pad group is a set of pads. For example, an **oscillator pad** usually comprises a set of two adjacent pads that we connect to an external crystal. The oscillator circuit and the two signal pads form a single logic cell. Another common example is a **clock pad**. Some foundries allow a special form of **corner pad** (normal

pads are **edge pads**) that squeezes two pads into the area at the corners of a chip using a special **two-pad corner cell**, to help meet **bond-wire angle design rules** (see also Figure 16.13b and c).

To reduce the series resistive and inductive impedance of power supply networks, it is normal to use multiple VDD and VSS pads. This is particularly important with the **simultaneously switching outputs** (**SSOs**) that occur when driving buses off-chip [Wada, Eino, and Anami, 1990]. The output pads can easily consume most of the power on a CMOS ASIC, because the load on a pad (usually tens of picofarads) is much larger than typical on-chip capacitive loads. Depending on the technology it may be necessary to provide dedicated VDD and VSS pads for every few SSOs. Design rules set how many SSOs can be used per VDD/VSS pad pair. These dedicated VDD/VSS pads must "follow" groups of output pads as they are seeded or planned on the floorplan. With some chip packages this can become difficult because design rules limit the location of package pins that may be used for supplies (due to the differing series inductance of each pin).

Using a **pad mapping** we translate the **logical pad** in a netlist to a **physical pad** from a **pad library**. We might control pad seeding and mapping in the floorplanner. The handling of I/O pads can become quite complex; there are several nonobvious factors that must be considered when generating a pad ring:

- Ideally we would only need to design library pad cells for one orientation. For example, an edge pad for the south side of the chip, and a corner pad for the southeast corner. We could then generate other orientations by rotation and flipping (mirroring). Some ASIC vendors will not allow rotation or mirroring of logic cells in the mask file. To avoid these problems we may need to have separate horizontal, vertical, left-handed, and right-handed pad cells in the library with appropriate logical to physical pad mappings.
- If we mix pad-limited and core-limited edge pads in the same pad ring, this complicates the design of corner pads. Usually the two types of edge pad cannot abut. In this case a corner pad also becomes a **pad-format changer**, or **hybrid corner pad**.
- In single-supply chips we have one VDD net and one VSS net, both **global power nets**. It is also possible to use **mixed power supplies** (for example, 3.3 V and 5 V) or **multiple power supplies** (digital VDD, analog VDD).

Figure 16.13(a) and (b) are magnified views of the southeast corner of our example chip and show the different types of I/O cells. Figure 16.13(c) shows a **stagger-bond** arrangement using two rows of I/O pads. In this case the design rules for bond wires (the spacing and the angle at which the bond wires leave the pads) become very important.

Figure 16.13(d) shows an **area-bump** bonding arrangement (also known as flip-chip, solder-bump or C4, terms coined by IBM who developed this technology [Masleid, 1991]) used, for example, with **ball-grid array** (**BGA**) packages. Even

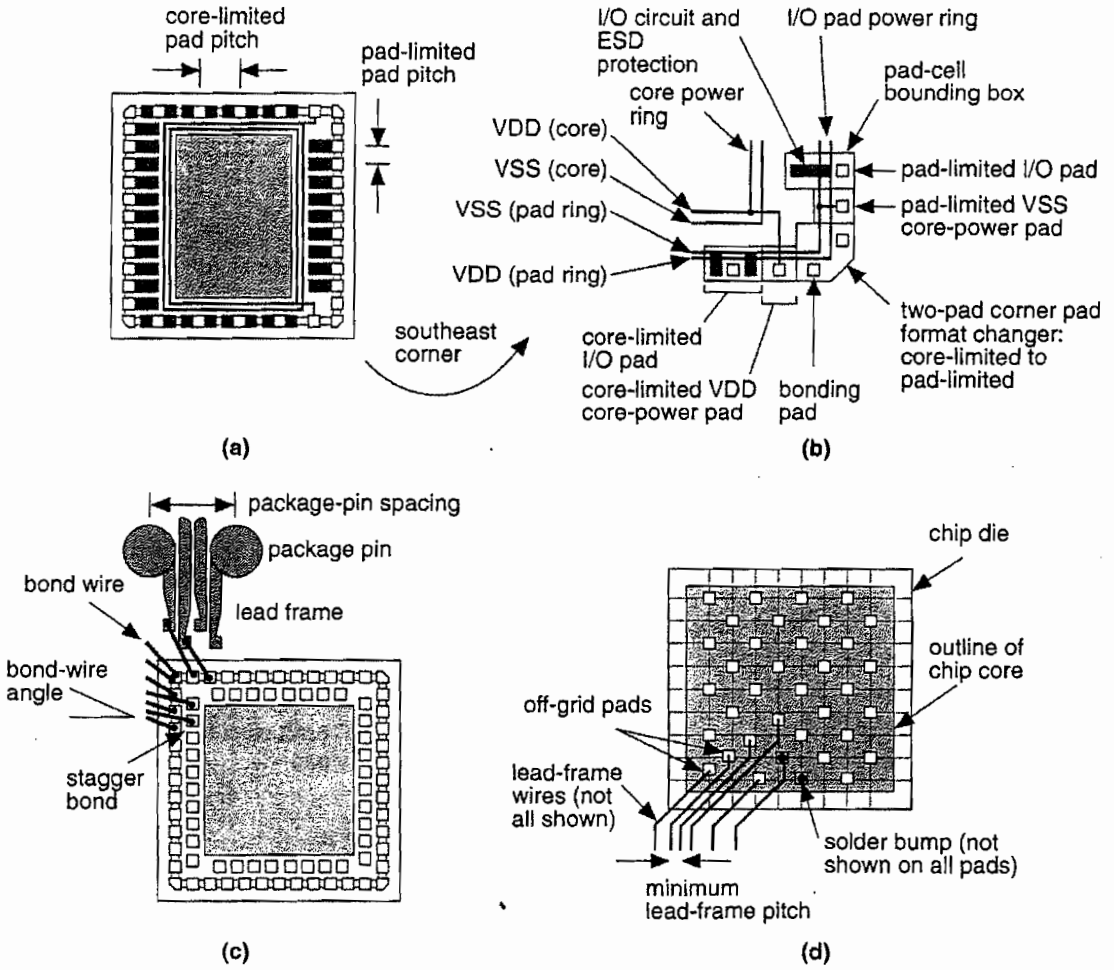


FIGURE 16.13 Bonding pads. (a) This chip uses both pad-limited and core-limited pads. (b) A hybrid corner pad. (c) A chip with stagger-bonded pads. (d) An area-bump bonded chip (or flip-chip). The chip is turned upside down and solder bumps connect the pads to the lead frame.

though the bonding pads are located in the center of the chip, the I/O circuits are still often located at the edges of the chip because of difficulties in power supply distribution and integrating I/O circuits together with logic in the center of the die.

In an MGA the pad spacing and I/O-cell spacing is fixed—each pad occupies a fixed **pad slot** (or **pad site**). This means that the properties of the pad I/O are also fixed but, if we need to, we can parallel adjacent output cells to increase the drive. To increase flexibility further the I/O cells can use a separation, the **I/O-cell pitch**, that is smaller than the **pad pitch**. For example, three 4 mA driver cells can occupy two pad slots. Then we can use two 4 mA output cells in parallel to drive one pad, forming an 8 mA output pad as shown in Figure 16.14. This arrangement also means the I/O pad cells can be changed without changing the base array. This is useful as bonding techniques improve and the pads can be moved closer together.

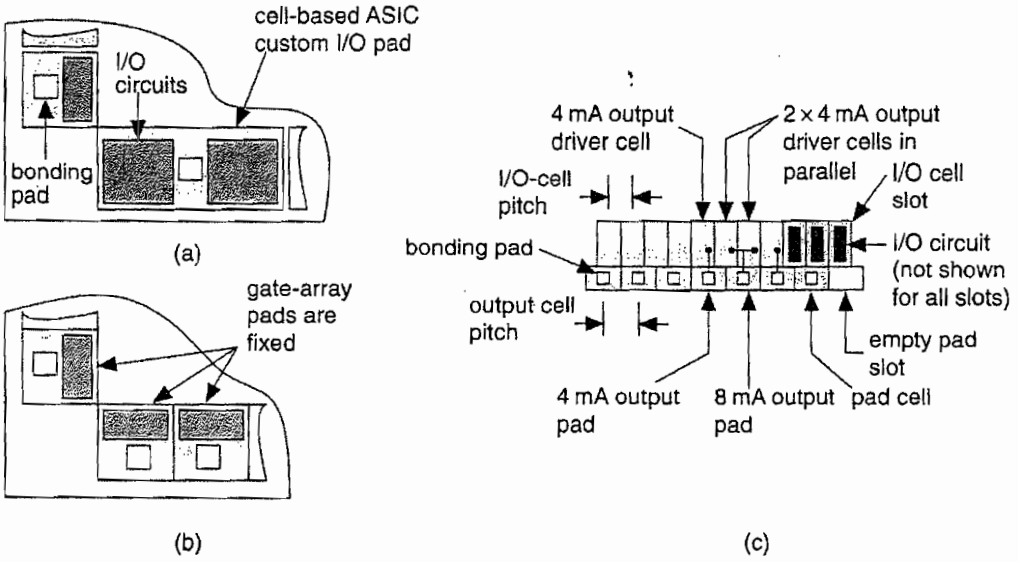


FIGURE 16.14 Gate-array I/O pads. (a) Cell-based ASICs may contain pad cells of different sizes and widths. (b) A corner of a gate-array base. (c) A gate-array base with different I/O cell and pad pitches.

Figure 16.15 shows two possible power distribution schemes. The long direction of a rectangular channel is the **channel spine**. Some automatic routers may require that metal lines parallel to a channel spine use a **preferred layer** (either m1, m2, or m3). Alternatively we say that a particular metal layer runs in a **preferred direction**. Since we can have both horizontal and vertical channels, we may have the situation shown in Figure 16.15, where we have to decide whether to use a preferred layer or the preferred direction for some channels. This may or may not be handled automatically by the routing software.

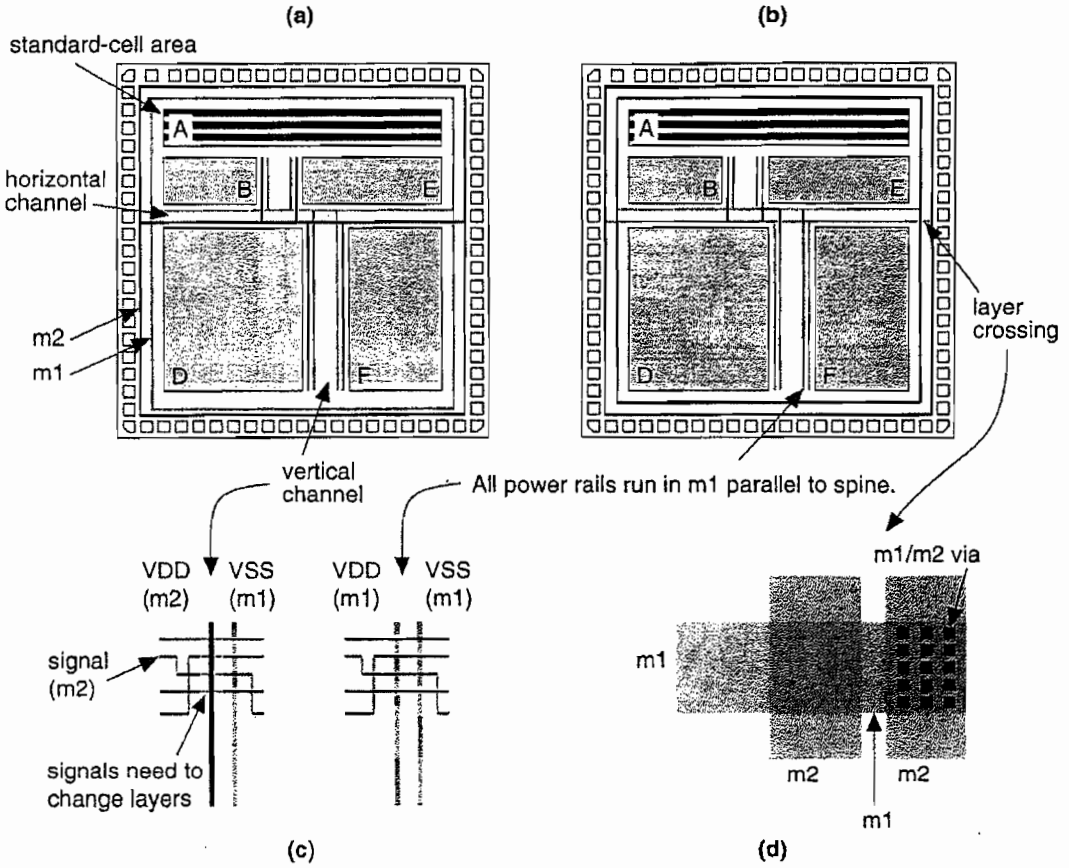


FIGURE 16.15 Power distribution. (a) Power distributed using m1 for VSS and m2 for VDD. This helps minimize the number of vias and layer crossings needed but causes problems in the routing channels. (b) In this floorplan m1 is run parallel to the longest side of all channels, the channel spine. This can make automatic routing easier but may increase the number of vias and layer crossings. (c) An expanded view of part of a channel (interconnect is shown as lines). If power runs on different layers along the spine of a channel, this forces signals to change layers. (d) A closeup of VDD and VSS buses as they cross. Changing layers requires a large number of via contacts to reduce resistance.

16.1.6 Clock Planning

Figure 16.16(a) shows a **clock spine** (not to be confused with a channel spine) routing scheme with all clock pins driven directly from the clock driver. MGAs and FPGAs often use this fish bone type of clock distribution scheme. Figure 16.16(b)

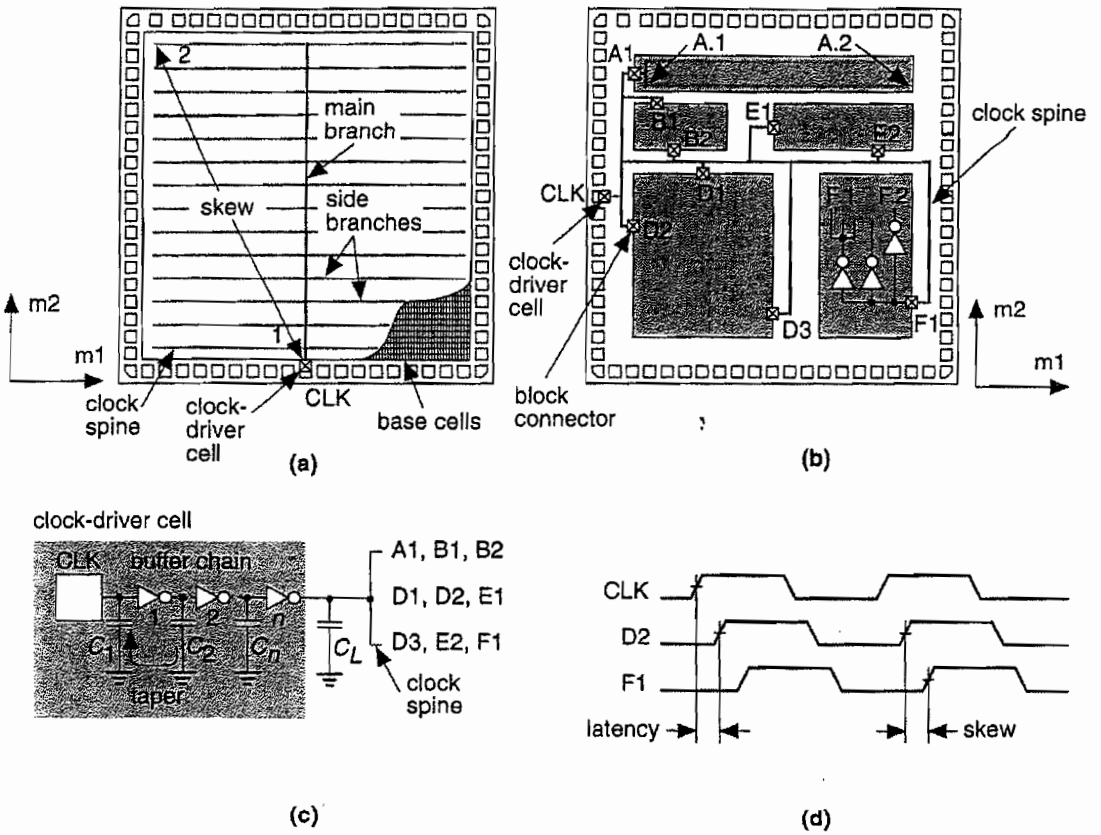


FIGURE 16.16 Clock distribution. (a) A clock spine for a gate array. (b) A clock spine for a cell-based ASIC (typical chips have thousands of clock nets). (c) A clock spine is usually driven from one or more clock-driver cells. Delay in the driver cell is a function of the number of stages and the ratio of output to input capacitance for each stage (taper). (d) Clock latency and clock skew. We would like to minimize both latency and skew.

shows a clock spine for a cell-based ASIC. Figure 16.16(c) shows the clock-driver cell, often part of a special clock-pad cell. Figure 16.16(d) illustrates **clock skew** and **clock latency**. Since all clocked elements are driven from one net with a clock spine, skew is caused by differing interconnect lengths and loads. If the clock-driver delay is much larger than the interconnect delays, a clock spine achieves minimum skew but with long latency.

Clock skew represents a fraction of the clock period that we cannot use for computation. A clock skew of 500 ps with a 200 MHz clock means that we waste 500 ps of every 5 ns clock cycle, or 10 percent of performance. Latency can cause a similar loss of performance at the system level when we need to resynchronize our output signals with a master system clock.

Figure 16.16(c) illustrates the construction of a clock-driver cell. The delay through a chain of CMOS gates is minimized when the ratio between the input capacitance C_1 and the output (load) capacitance C_2 is about 3 (exactly $e \approx 2.7$, an exponential ratio, if we neglect the effect of parasitics). This means that the fastest way to drive a large load is to use a chain of buffers with their input and output loads chosen to maintain this ratio, or **taper** (we use this as a noun and a verb). This is not necessarily the smallest or lowest-power method, though.

Suppose we have an ASIC with the following specifications:

- 40,000 flip-flops
- Input capacitance of the clock input to each flip-flop is 0.025 pF
- Clock frequency is 200 MHz
- $V_{DD} = 3.3$ V
- Chip size is 20 mm on a side
- Clock spine consists of 200 lines across the chip
- Interconnect capacitance is 2 $pFcm^{-1}$

In this case the clock-spine capacitance $C_L = 200 \times 2\ cm \times 2\ pFcm^{-1} = 800$ pF. If we drive the clock spine with a chain of buffers with taper equal to $e \approx 2.7$, and with a first-stage input capacitance of 0.025 pF (a reasonable value for a 0.5 μm process), we will need

$$\log \frac{800 \times 10^{-12}}{0.025 \times 10^{-12}} = 10.4, \quad \text{or 11 stages.} \tag{16.1}$$

The power dissipated charging the input capacitance of the flip-flop clock is fCV^2 or

$$P_1^1 = (4 \times 10^4)\ (200\ \text{MHz})\ (0.025\ \text{pF})\ (3.3\ \text{V})^2 = 2.178\ \text{W}, \tag{16.2}$$

or approximately 2 W. This is only a little larger than the power dissipated driving the 800 pF clock-spine interconnect that we can calculate as follows:

$$P_1^2 = (200)\ (200\ \text{MHz})\ (20\ \text{mm})\ (2\ pFcm^{-1})\ (3.3\ \text{V})^2 = 1.7424\ \text{W}. \tag{16.3}$$

All of this power is dissipated in the clock-driver cell. The worst problem, however, is the enormous peak current in the final inverter stage. If we assume the needed rise time is 0.1 ns (with a 200 MHz clock whose period is 5 ns), the peak current would have to approach

$$I = \frac{(800\ \text{pF})\ (\ 3.3\text{V})}{0.1\ \text{ns}} = 25\ \text{A}. \tag{16.4}$$

Clearly such a current is not possible without extraordinary design techniques. Clock spines are used to drive loads of 100–200 pF but, as is apparent from the power dissipation problems of this example, it would be better to find a way to spread the power dissipation more evenly across the chip.

We can design a tree of clock buffers so that the taper of each stage is $e \approx 2.7$ by using a fanout of three at each node, as shown in Figure 16.17(a) and (b). The **clock tree**, shown in Figure 16.17(c), uses the same number of stages as a clock spine, but with a lower peak current for the inverter buffers. Figure 16.17(c) illustrates that we now have another problem—we need to balance the delays through the tree carefully to minimize clock skew (see Section 17.3.1, "Clock Routing").

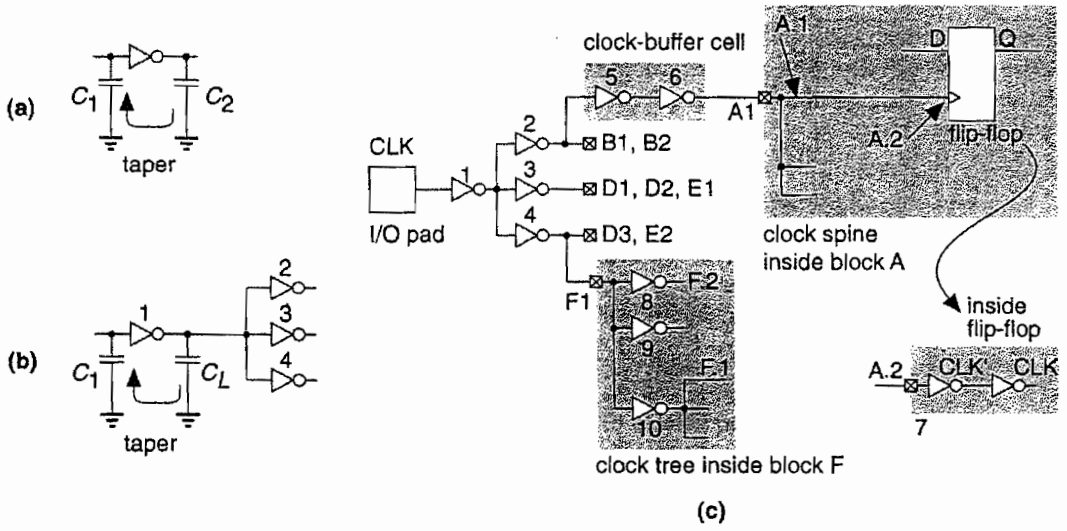


FIGURE 16.17 A clock tree. (a) Minimum delay is achieved when the taper of successive stages is about 3. (b) Using a fanout of three at successive nodes. (c) A clock tree for the cell-based ASIC of Figure 16.16b. We have to balance the clock arrival times at all of the leaf nodes to minimize clock skew.

Designing a clock tree that balances the rise and fall times at the leaf nodes has the beneficial side-effect of minimizing the effect of **hot-electron wearout**. This problem occurs when an electron gains enough energy to become "hot" and jump out of the channel into the gate oxide (the problem is worse for electrons in *n*-channel devices because electrons are more mobile than holes). The trapped electrons change the threshold voltage of the device and this alters the delay of the buffers. As the buffer delays change with time, this introduces unpredictable skew. The problem is worst when the *n*-channel device is carrying maximum current with a high voltage across the channel—this occurs during the rise-and fall-time transitions. Balancing the rise and fall times in each buffer means that they all wear out at the same rate, minimizing any additional skew.

A **phase-locked loop** (PLL) is an electronic flywheel that locks in frequency to an input clock signal. The input and output frequencies may differ in phase, however. This means that we can, for example, drive a clock network with a PLL in such a way that the output of the clock network is locked in phase to the incoming clock, thus eliminating the latency of the clock network. A PLL can also help to reduce random variation of the input clock frequency, known as **jitter**, which, since it is unpredictable, must also be discounted from the time available for computation in each clock cycle. Actel was one of the first FPGA vendors to incorporate PLLs, and Actel's online product literature explains their use in ASIC design.

16.2 Placement

After completing a floorplan we can begin placement of the logic cells within the flexible blocks. Placement is much more suited to automation than floorplanning. Thus we shall need measurement techniques and algorithms. After we complete floorplanning and placement, we can predict both intrablock and interblock capacitances. This allows us to return to logic synthesis with more accurate estimates of the capacitive loads that each logic cell must drive.

16.2.1 Placement Terms and Definitions

CBIC, MGA, and FPGA architectures all have rows of logic cells separated by the interconnect—these are **row-based ASICs**. Figure 16.18 shows an example of the interconnect structure for a CBIC. Interconnect runs in horizontal and vertical directions in the channels and in the vertical direction by crossing through the logic cells. Figure 16.18(c) illustrates the fact that it is possible to use **over-the-cell routing** (**OTC** routing) in areas that are not blocked. However, OTC routing is complicated by the fact that the logic cells themselves may contain metal on the routing layers. We shall return to this topic in Section 17.2.7, "Multilevel Routing." Figure 16.19 shows the interconnect structure of a two-level metal MGA.

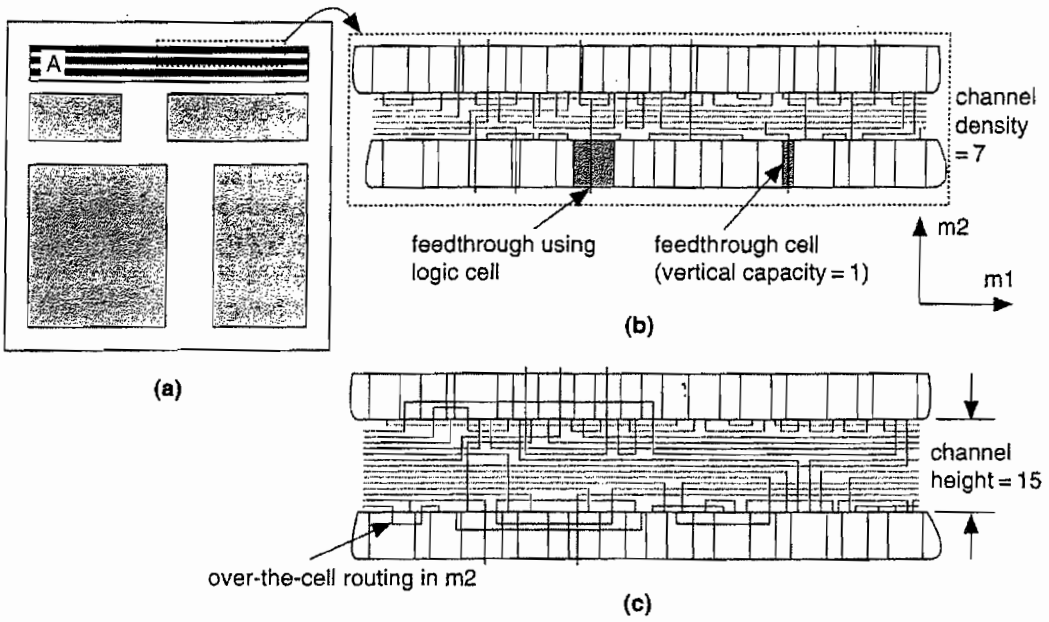


FIGURE 16.18 Interconnect structure. (a) The two-level metal CBIC floorplan shown in Figure 16.11b. (b) A channel from the flexible block A. This channel has a channel height equal to the maximum channel density of 7 (there is room for seven interconnects to run horizontally in m1). (c) A channel that uses OTC (over-the-cell) routing in m2.

Most ASICs currently use two or three levels of metal for signal routing. With two layers of metal, we route within the rectangular channels using the first metal layer for horizontal routing, parallel to the channel spine, and the second metal layer for the vertical direction (if there is a third metal layer it will normally run in the horizontal direction again). The maximum number of horizontal interconnects that can be placed side by side, parallel to the channel spine, is the **channel capacity**.

Vertical interconnect uses **feedthroughs** (or **feedthrus** in the United States) to cross the logic cells. Here are some commonly used terms with explanations (there are no generally accepted definitions):

- An unused **vertical track** (or just **track**) in a logic cell is called an **uncommitted feedthrough** (also **built-in feedthrough**, **implicit feedthrough**, or **jumper**).
- A vertical strip of metal that runs from the top to bottom of a cell (for **double-entry cells**), but has no connections inside the cell, is also called a feedthrough or jumper.

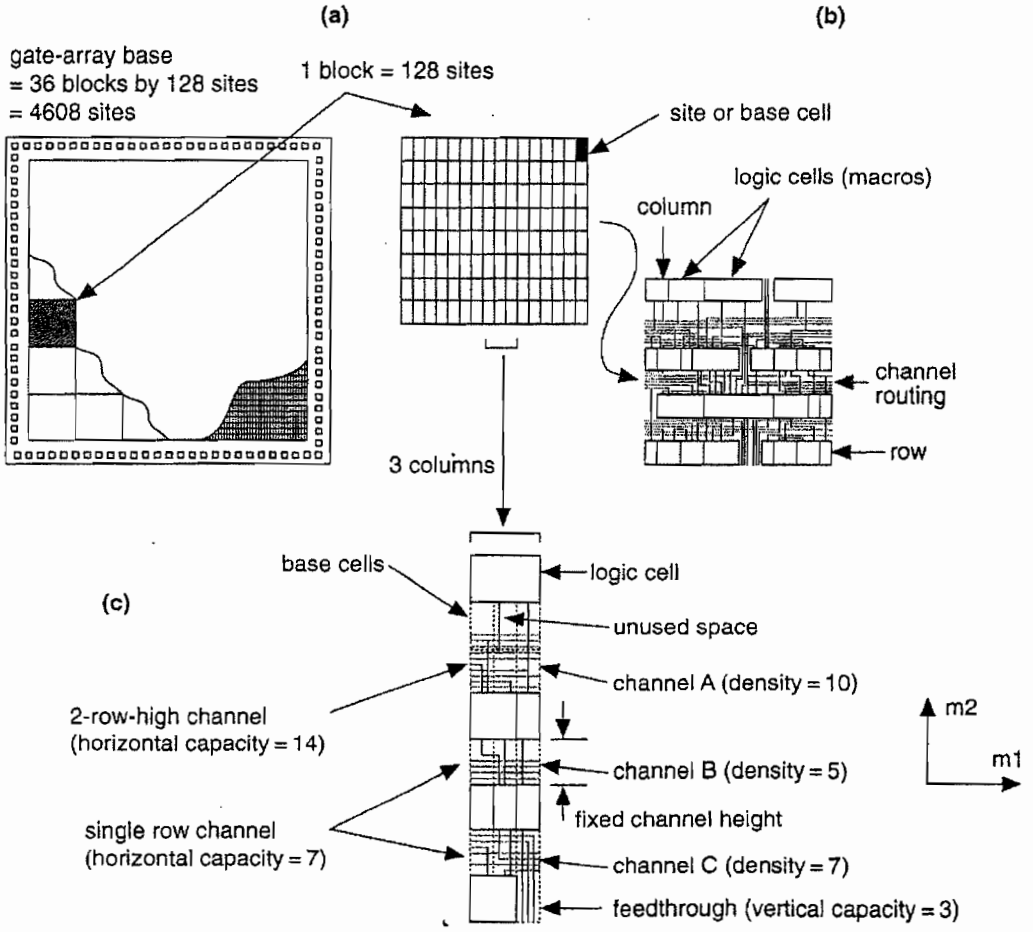


FIGURE 16.19 Gate-array interconnect. (a) A small two-level metal gate array (about 4.6 k-gate). (b) Routing in a block. (c) Channel routing showing channel density and channel capacity. The channel height on a gate array may only be increased in increments of a row. If the interconnect does not use up all of the channel, the rest of the space is wasted. The interconnect in the channel runs in m1 in the horizontal direction with m2 in the vertical direction.

- Two connectors for the same physical net are **electrically equivalent connectors** (or **equipotential connectors**). For double-entry cells these are usually at the top and bottom of the logic cell.
- A dedicated **feedthrough cell** (or **crosser cell**) is an empty cell (with no logic) that can hold one or more vertical interconnects. These are used if there are no other feedthroughs available.

- A **feedthrough pin** or **feedthrough terminal** is an input or output that has connections at both the top and bottom of the standard cell.
- A **spacer cell** (usually the same as a feedthrough cell) is used to fill space in rows so that the ends of all rows in a flexible block may be aligned to connect to power buses, for example.

There is no standard terminology for connectors and the terms can be very confusing. There is a difference between connectors that are joined inside the logic cell using a high-resistance material such as polysilicon and connectors that are joined by low-resistance metal. The high-resistance kind are really two separate **alternative connectors** (that cannot be used as a feedthrough), whereas the low-resistance kind are electrically equivalent connectors. There may be two or more connectors to a logic cell, which are not joined inside the cell, and which must be joined by the router (**must-join connectors**).

There are also **logically equivalent connectors** (or functionally equivalent connectors, sometimes also called just equivalent connectors—which is very confusing). The two inputs of a two-input NAND gate may be logically equivalent connectors. The placement tool can swap these without altering the logic (but the two inputs may have different delay properties, so it is not always a good idea to swap them). There can also be **logically equivalent connector groups**. For example, in an OAI22 (OR-AND-INVERT) gate there are four inputs: A1, A2 are inputs to one OR gate (gate A), and B1, B2 are inputs to the second OR gate (gate B). Then group A = (A1, A2) is logically equivalent to group B = (B1, B2)—if we swap one input (A1 or A2) from gate A to gate B, we must swap the other input in the group (A2 or A1).

In the case of channeled gate arrays and FPGAs, the horizontal interconnect areas—the channels, usually on m1—have a fixed capacity (sometimes they are called **fixed-resource ASICs** for this reason). The channel capacity of CBICs and channelless MGAs can be expanded to hold as many interconnects as are needed. Normally we choose, as an objective, to minimize the number of interconnects that use each channel. In the vertical interconnect direction, usually m2, FPGAs still have fixed resources. In contrast the placement tool can always add vertical feedthroughs to a channeled MGA, channelless MGA, or CBIC. These problems become less important as we move to three and more levels of interconnect.

16.2.2 Placement Goals and Objectives

The goal of a placement tool is to arrange all the logic cells within the flexible blocks on a chip. Ideally, the objectives of the placement step are to

- Guarantee the router can complete the routing step
- Minimize all the critical net delays
- Make the chip as dense as possible

We may also have the following additional objectives:

- Minimize power dissipation
- Minimize cross talk between signals

Objectives such as these are difficult to define in a way that can be solved with an algorithm and even harder to actually meet. Current placement tools use more specific and achievable criteria. The most commonly used placement objectives are one or more of the following:

- Minimize the total estimated interconnect length
- Meet the timing requirements for critical nets
- Minimize the interconnect congestion

Each of these objectives in some way represents a compromise.

16.2.3 Measurement of Placement Goals and Objectives

In order to determine the quality of a placement, we need to be able to measure it. We need an approximate measure of interconnect length, closely correlated with the final interconnect length, that is easy to calculate.

The graph structures that correspond to making all the connections for a net are known as **trees on graphs** (or just **trees**). Special classes of trees—**Steiner trees**—minimize the total length of interconnect and they are central to ASIC routing algorithms. Figure 16.20 shows a minimum Steiner tree. This type of tree uses diagonal connections—we want to solve a restricted version of this problem, using interconnects on a rectangular grid. This is called **rectilinear routing** or **Manhattan routing** (because of the east–west and north–south grid of streets in Manhattan). We say that the **Euclidean distance** between two points is the straight-line distance ("as the crow flies"). The **Manhattan distance** (or rectangular distance) between two points is the distance we would have to walk in New York.

The **minimum rectilinear Steiner tree** (MRST) is the shortest interconnect using a rectangular grid. The determination of the MRST is in general an NP-complete problem—which means it is hard to solve. For small numbers of terminals heuristic algorithms do exist, but they are expensive to compute. Fortunately we only need to estimate the length of the interconnect. Two approximations to the MRST are shown in Figure 16.21.

The **complete graph** has connections from each terminal to every other terminal [Hanan, Wolff, and Agule, 1973]. The **complete-graph measure** adds all the interconnect lengths of the complete-graph connection together and then divides by $n/2$, where n is the number of terminals. We can justify this since, in a graph with n terminals, $(n-1)$ interconnects will emanate from each terminal to join the other $(n-1)$ terminals in a complete graph connection. That makes $n(n-1)$ interconnects in total. However, we have then made each connection twice. So there are one-half

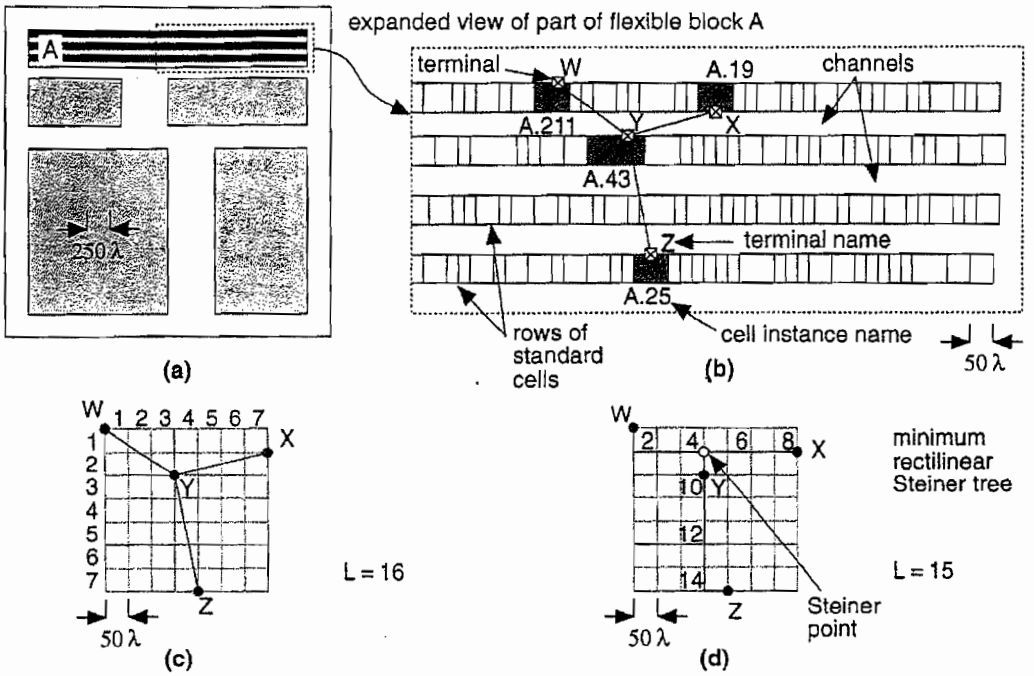


FIGURE 16.20 Placement using trees on graphs. (a) The floorplan from Figure 16.11b. (b) An expanded view of the flexible block A showing four rows of standard cells for placement (typical blocks may contain thousands or tens of thousands of logic cells). We want to find the length of the net shown with four terminals, W through Z, given the placement of four logic cells (labeled: A.211, A.19, A.43, A.25). (c) The problem for net (W, X, Y, Z) drawn as a graph. The shortest connection is the minimum Steiner tree. (d) The minimum rectilinear Steiner tree using Manhattan routing. The rectangular (Manhattan) interconnect-length measures are shown for each tree.

this many, or $n(n-1)/2$, interconnects needed for a complete graph connection. Now we actually only need $(n-1)$ interconnects to join n terminals, so we have $n/2$ times as many interconnects as we really need. Hence we divide the total net length of the complete graph connection by $n/2$ to obtain a more reasonable estimate of minimum interconnect length. Figure 16.21(a) shows an example of the complete-graph measure.

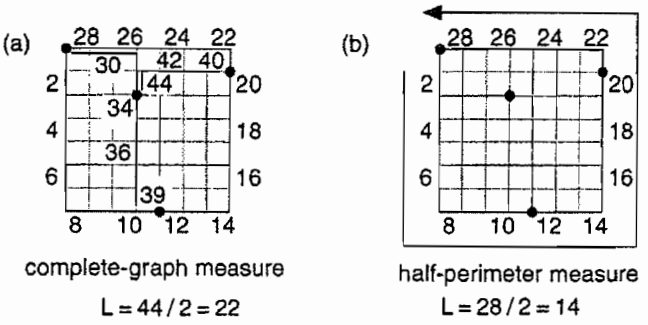


FIGURE 16.21 Interconnect-length measures. (a) Complete-graph measure. (b) Half-perimeter measure.

The **bounding box** is the smallest rectangle that encloses all the terminals (not to be confused with a logic cell bounding box, which encloses all the layout in a logic cell). The **half-perimeter measure** (or bounding-box measure) is one-half the perimeter of the bounding box (Figure 16.21b) [Schweikert, 1976]. For nets with two or three terminals (corresponding to a fanout of one or two, which usually includes over 50 percent of all nets on a chip), the half-perimeter measure is the same as the minimum Steiner tree. For nets with four or five terminals, the minimum Steiner tree is between one and two times the half-perimeter measure [Hanan, 1966]. For a circuit with m nets, using the half-perimeter measure corresponds to minimizing the cost function,

$$f = \frac{1}{2}\sum_{i=1}^{m} h_i \,, \tag{16.5}$$

where h_i is the half-perimeter measure for net i.

It does not really matter if our approximations are inaccurate if there is a good correlation between actual interconnect lengths (after routing) and our approximations. Figure 16.22 shows that we can adjust the complete-graph and half-perimeter measures using correction factors [Goto and Matsuda, 1986]. Now our wiring length approximations are functions, not just of the terminal positions, but also of the number of terminals, and the size of the bounding box. One practical example adjusts a Steiner-tree approximation using the number of terminals [Chao, Nequist, and Vuong, 1990]. This technique is used in the Cadence Gate Ensemble placement tool, for example.

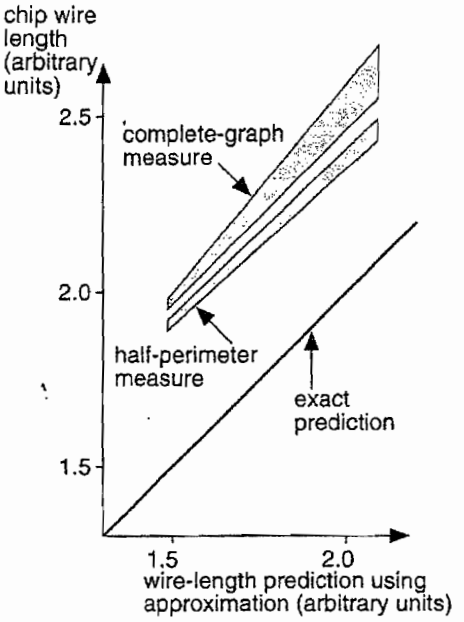


FIGURE 16.22 Correlation between total length of chip interconnect and the half-perimeter and complete-graph measures.

One problem with the measurements we have described is that the MRST may only approximate the interconnect that will be completed by the detailed router. Some programs have a **meander factor** that specifies, on average, the ratio of the interconnect created by the routing tool to the interconnect-length estimate used by the placement tool. Another problem is that we have concentrated on finding estimates to the MRST, but the MRST that minimizes total net length may not minimize net delay (see Section 16.2.8).

There is no point in minimizing the interconnect length if we create a placement that is too congested to route. If we use minimum **interconnect congestion** as an additional placement objective, we need some way of measuring it. What we are trying to measure is interconnect density. Unfortunately we always use the term *density* to mean channel density (which we shall discuss in Section 17.2.2, "Measurement of Channel Density"). In this chapter, while we are discussing placement, we shall try to use the term *congestion*, instead of density, to avoid any confusion.

One measure of interconnect congestion uses the **maximum cut line**. Imagine a horizontal or vertical line drawn anywhere across a chip or block, as shown in Figure 16.23. The number of interconnects that must cross this line is the **cut size** (the number of interconnects we cut). The maximum cut line has the highest cut size.

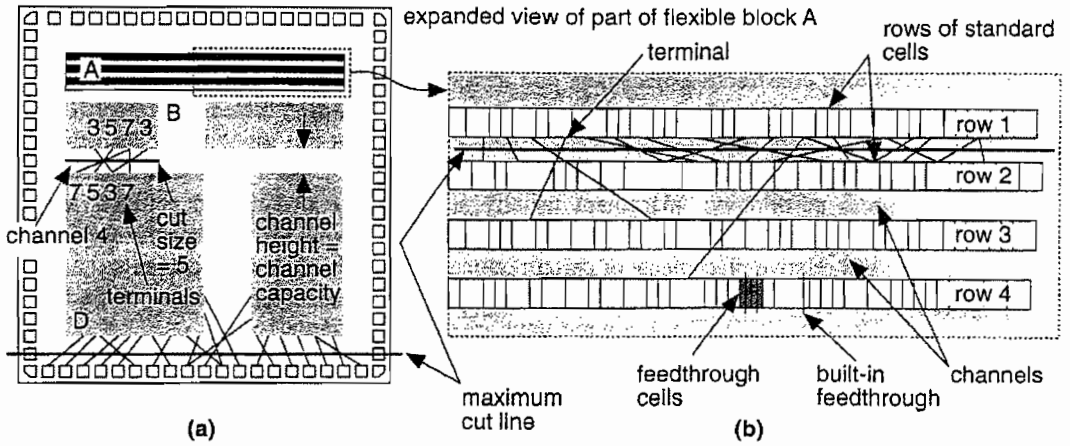


FIGURE 16.23 Interconnect congestion for the cell-based ASIC from Figure 16.11(b). (a) Measurement of congestion. (b) An expanded view of flexible block A shows a maximum cut line.

Many placement tools minimize estimated interconnect length or interconnect congestion as objectives. The problem with this approach is that a logic cell may be placed a long way from another logic cell to which it has just one connection. This logic cell with one connection is less important as far as the total wire length is concerned than other logic cells, to which there are many connections. However, the one long connection may be critical as far as timing delay is concerned. As technology is scaled, interconnection delays become larger relative to circuit delays and this problem gets worse.

In **timing-driven placement** we must estimate delay for every net for every trial placement, possibly for hundreds of thousands of gates. We cannot afford to use anything other than the very simplest estimates of net delay. Unfortunately, the minimum-length Steiner tree does not necessarily correspond to the interconnect path that minimizes delay. To construct a minimum-delay path we may have to route with non-Steiner trees. In the placement phase typically we take a simple interconnect-length approximation to this minimum-delay path (typically the half-perimeter measure). Even when we can estimate the length of the interconnect, we do not yet have information on which layers and how many vias the interconnect will use or how wide it will be. Some tools allow us to include estimates for these parameters. Often we can specify **metal usage**, the percentage of routing on the different layers to expect from the router. This allows the placement tool to estimate RC values and delays—and thus minimize delay.

16.2.4 Placement Algorithms

There are two classes of placement algorithms commonly used in commercial CAD tools: constructive placement and iterative placement improvement. A **constructive placement method** uses a set of rules to arrive at a constructed placement. The most commonly used methods are variations on the **min-cut algorithm**. The other commonly used constructive placement algorithm is the **eigenvalue method**. As in system partitioning, placement usually starts with a constructed solution and then improves it using an iterative algorithm. In most tools we can specify the locations and relative placements of certain critical logic cells as **seed placements**.

The **min-cut placement** method uses successive application of partitioning [Breuer, 1977]. The following steps are shown in Figure 16.24:

1. Cut the placement area into two pieces.
2. Swap the logic cells to minimize the cut cost.
3. Repeat the process from step 1, cutting smaller pieces until all the logic cells are placed.

Usually we divide the placement area into **bins**. The size of a bin can vary, from a bin size equal to the base cell (for a gate array) to a bin size that would hold several logic cells. We can start with a large bin size, to get a rough placement, and then reduce the bin size to get a final placement.

The **eigenvalue placement algorithm** uses the cost matrix or weighted **connectivity matrix** (eigenvalue methods are also known as **spectral methods**) [Hall, 1970]. The measure we use is a cost function f that we shall minimize, given by

$$f = \frac{1}{2} \sum_{i,j=1}^{n} c_{ij} \, d_{ij}^{\,2}, \tag{16.6}$$

where $\mathbf{C} = [c_{ij}]$ is the (possibly weighted) connectivity matrix, and d_{ij} is the Euclidean distance between the centers of logic cell i and logic cell j. Since we are going to minimize a cost function that is the square of the distance between logic cells, these methods are also known as **quadratic placement** methods. This type of cost function leads to a simple mathematical solution. We can rewrite the cost function f in matrix form:

$$f = \frac{1}{2} \sum_{i,j=1}^{n} c_{ij} (x_i - x_j)^2 + (y_i - y_j)^2 = \mathbf{x}^T \mathbf{B} \mathbf{x} + \mathbf{y}^T \mathbf{B} \mathbf{y} \, , \tag{16.7}$$

In Eq. 16.7, **B** is a symmetric matrix, the **disconnection matrix** (also called the Laplacian).

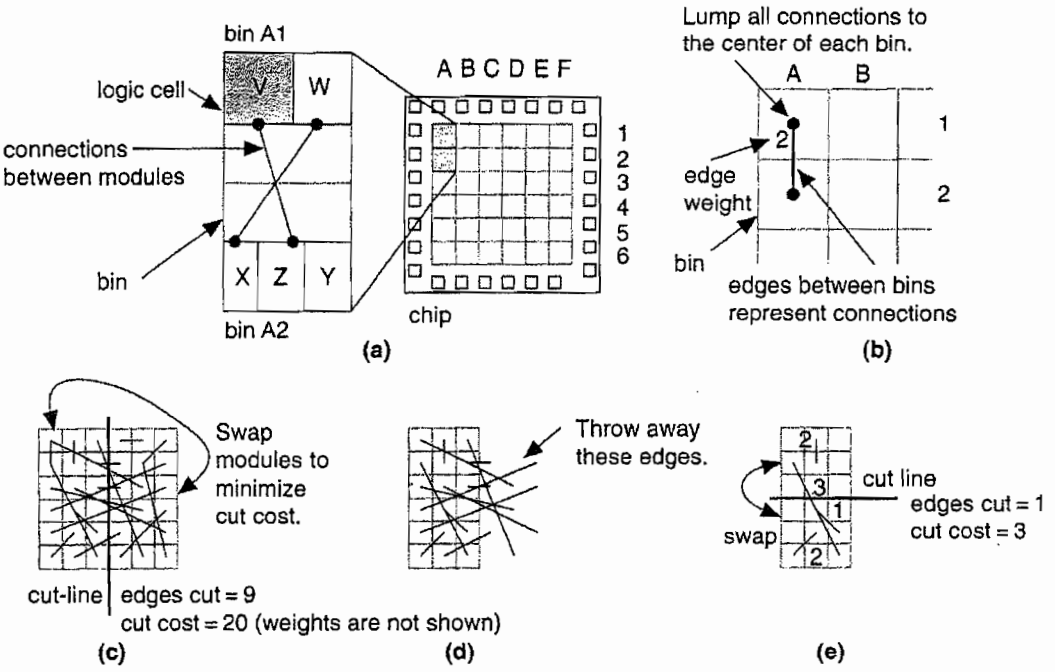


FIGURE 16.24 Min-cut placement. (a) Divide the chip into bins using a grid. (b) Merge all connections to the center of each bin. (c) Make a cut and swap logic cells between bins to minimize the cost of the cut. (d) Take the cut pieces and throw out all the edges that are not inside the piece. (e) Repeat the process with a new cut and continue until we reach the individual bins.

We may express the Laplacian **B** in terms of the connectivity matrix **C**; and **D**, a diagonal matrix (known as the degree matrix), defined as follows:

$$\mathbf{B} = \mathbf{D} - \mathbf{C};\ d_{ii} = \sum_{j=1}^{n} c_{ij},\ i = 1, \ldots, n;\ d_{ij} = 0,\ \ i \neq j. \tag{16.8}$$

We can simplify the problem by noticing that it is symmetric in the x- and y-coordinates. Let us solve the simpler problem of minimizing the cost function for the placement of logic cells along just the x-axis first. We can then apply this solution to the more general two-dimensional placement problem. Before we solve this simpler problem, we introduce a constraint that the coordinates of the logic cells must correspond to valid positions (the cells do not overlap and they are placed on-

grid). We make another simplifying assumption that all logic cells are the same size and we must place them in fixed positions. We can define a vector **p** consisting of the valid positions:

$$\mathbf{p} = [p_1, \ldots, p_n] . \tag{16.9}$$

For a valid placement the x-coordinates of the logic cells,

$$\mathbf{x} = [x_1, \ldots, x_n] , \tag{16.10}$$

must be a permutation of the fixed positions, **p**. We can show that requiring the logic cells to be in fixed positions in this way leads to a series of n equations restricting the values of the logic cell coordinates [Cheng and Kuh, 1984]. If we impose all of these constraint equations the problem becomes very complex. Instead we choose just one of the equations:

$$\sum_{i=1}^{n} x_i^2 = \sum_{i=1}^{n} p_i^2 . \tag{16.11}$$

Simplifying the problem in this way will lead to an approximate solution to the placement problem. We can write this single constraint on the x-coordinates in matrix form:

$$\mathbf{x}^T\mathbf{x} = \mathrm{P}; \qquad \mathrm{P} = \sum_{i=1}^{n} p_i^2 , \tag{16.12}$$

where P is a constant. We can now summarize the formulation of the problem, with the simplifications that we have made, for a one-dimensional solution. We must minimize a cost function, g (analogous to the cost function f that we defined for the two-dimensional problem in Eq. 16.7), where

$$g = \mathbf{x}^T\mathbf{B}\mathbf{x} \tag{16.13}$$

subject to the constraint:

$$\mathbf{x}^T\mathbf{x} = \mathrm{P} . \tag{16.14}$$

This is a standard problem that we can solve using a Lagrangian multiplier:

$$\Lambda = \mathbf{x}^T\mathbf{B}\mathbf{x} - \lambda[\mathbf{x}^T\mathbf{x} - \mathrm{P}] . \tag{16.15}$$

To find the value of **x** that minimizes g we differentiate Λ partially with respect to **x** and set the result equal to zero. We get the following equation:

$$[\mathbf{B} - \lambda\mathbf{I}]\,\mathbf{x} = \mathbf{0} . \tag{16.16}$$

This last equation is called the **characteristic equation** for the disconnection matrix **B** and occurs frequently in matrix algebra (this λ has nothing to do with scaling). The solutions to this equation are the **eigenvectors** and **eigenvalues** of **B**. Multiplying Eq. 16.16 by $\mathbf{x}^T$ we get:

$$\lambda \mathbf{x}^T \mathbf{x} = \mathbf{x}^T \mathbf{B} \mathbf{x}. \tag{16.17}$$

However, since we imposed the constraint $\mathbf{x}^T\mathbf{x} = P$ and $\mathbf{x}^T\mathbf{B}\mathbf{x} = g$, then

$$\lambda = \frac{g}{P}. \tag{16.18}$$

The eigenvectors of the disconnection matrix **B** are the solutions to our placement problem. It turns out that (because something called the rank of matrix **B** is $n-1$) there is a degenerate solution with all x-coordinates equal ($\lambda = 0$)—this makes some sense because putting all the logic cells on top of one another certainly minimizes the interconnect. The smallest, nonzero, eigenvalue and the corresponding eigenvector provides the solution that we want. In the two-dimensional placement problem, the x- and y-coordinates are given by the eigenvectors corresponding to the two smallest, nonzero, eigenvalues. (In the next section a simple example illustrates this mathematical derivation.)

16.2.5 Eigenvalue Placement Example

Consider the following connectivity matrix **C** and its disconnection matrix **B**, calculated from Eq. 16.8 [Hall, 1970]:

$$\mathbf{C} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix}; \quad \mathbf{B} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix} - \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & -1 \\ 0 & 2 & -1 & -1 \\ 0 & -1 & 1 & 0 \\ -1 & -1 & 0 & 2 \end{bmatrix} \tag{16.19}$$

Figure 16.25(a) shows the corresponding network with four logic cells (1–4) and three nets (A-C). Here is a MatLab script to find the eigenvalues and eigenvectors of **B**:

```
C=[0 0 0 1; 0 0 1 1; 0 1 0 0; 1 1 0 0]
D=[1 0 0 0; 0 2 0 0; 0 0 1 0; 0 0 0 2]
B=D-C
[X,D] = eig(B)
```

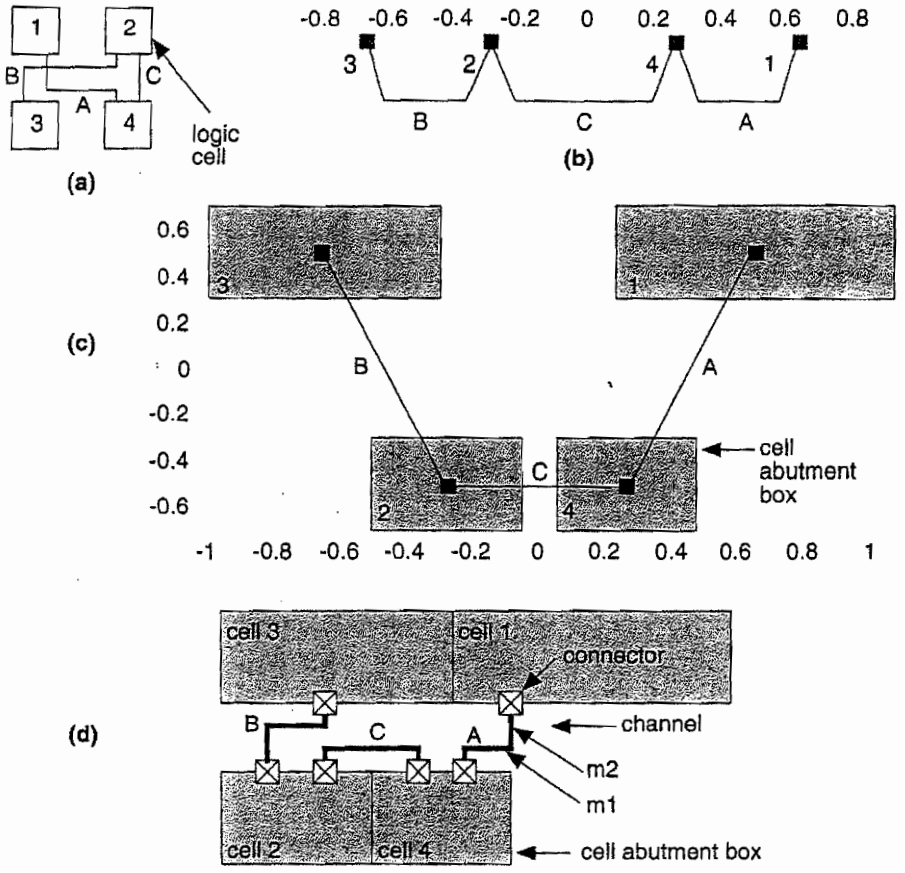


FIGURE 16.25 Eigenvalue placement. (a) An example network. (b) The one-dimensional placement.The small black squares represent the centers of the logic cells. (c) The two-dimensional placement. The eigenvalue method takes no account of the logic cell sizes or actual location of logic cell connectors. (d) A complete layout. We snap the logic cells to valid locations, leaving room for the routing in the channel.

Running this script, we find the eigenvalues of **B** are 0.5858, 0.0, 2.0, and 3.4142. The corresponding eigenvectors of **B** are

$$\begin{bmatrix} 0.6533 & 0.5000 & 0.5000 & -0.2706 \\ -0.2706 & 0.5000 & -0.5000 & -0.6533 \\ -0.6533 & 0.5000 & 0.5000 & 0.2706 \\ 0.2706 & 0.5000 & -0.5000 & 0.6533 \end{bmatrix} \tag{16.20}$$

For a one-dimensional placement (Figure 16.25b), we use the eigenvector (0.6533, –0.2706, –0.6533, –0.2706) corresponding to the smallest nonzero eigenvalue (which is 0.5858) to place the logic cells along the x-axis. The two-dimensional placement (Figure 16.25c) uses these same values for the x-coordinates and the eigenvector (0.5, –0.5, 0.5, –0.5) that corresponds to the next largest eigenvalue (which is 2.0) for the y-coordinates. Notice that the placement shown in Figure 16.25(c), which shows logic-cell outlines (the logic-cell abutment boxes), takes no account of the cell sizes, and cells may even overlap at this stage. This is because, in Eq. 16.11, we discarded all but one of the constraints necessary to ensure valid solutions. Often we use the approximate eigenvalue solution as an initial placement for one of the iterative improvement algorithms that we shall discuss in Section 16.2.6.

16.2.6 Iterative Placement Improvement

An **iterative placement improvement** algorithm takes an existing placement and tries to improve it by moving the logic cells. There are two parts to the algorithm:

- The selection criteria that decides which logic cells to try moving.
- The measurement criteria that decides whether to move the selected cells.

There are several **interchange** or **iterative exchange** methods that differ in their selection and measurement criteria:

- pairwise interchange,
- force-directed interchange,
- force-directed relaxation, and
- force-directed pairwise relaxation.

All of these methods usually consider only pairs of logic cells to be exchanged. A source logic cell is picked for trial exchange with a destination logic cell. We have already discussed the use of interchange methods applied to the system partitioning step. The most widely used methods use group migration, especially the Kernighan–Lin algorithm. The **pairwise-interchange algorithm** is similar to the interchange algorithm used for iterative improvement in the system partitioning step:

1. Select the source logic cell at random.
2. Try all the other logic cells in turn as the destination logic cell.
3. Use any of the measurement methods we have discussed to decide on whether to accept the interchange.
4. The process repeats from step 1, selecting each logic cell in turn as a source logic cell.

Figure 16.26(a) and (b) show how we can extend pairwise interchange to swap more than two logic cells at a time. If we swap λ logic cells at a time and find a locally optimum solution, we say that solution is **λ-optimum**. The **neighborhood exchange algorithm** is a modification to pairwise interchange that considers only

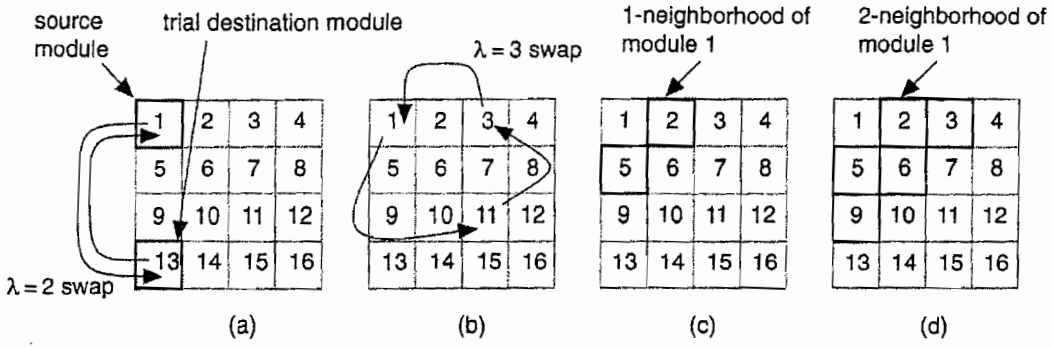


FIGURE 16.26 Interchange. (a) Swapping the source logic cell with a destination logic cell in pairwise interchange. (b) Sometimes we have to swap more than two logic cells at a time to reach an optimum placement, but this is expensive in computation time. Limiting the search to neighborhoods reduces the search time. Logic cells within a distance ε of a logic cell form an ε-neighborhood. (c) A one-neighborhood. (d) A two-neighborhood.

destination logic cells in a **neighborhood**—cells within a certain distance, ε, of the source logic cell. Limiting the search area for the destination logic cell to the **ε-neighborhood** reduces the search time. Figure 16.26(c) and (d) show the one- and two-neighborhoods (based on Manhattan distance) for a logic cell.

Neighborhoods are also used in some of the **force-directed placement methods**. Imagine identical springs connecting all the logic cells we wish to place. The number of springs is equal to the number of connections between logic cells. The effect of the springs is to pull connected logic cells together. The more highly connected the logic cells, the stronger the pull of the springs. The force on a logic cell i due to logic cell j is given by **Hooke's law**, which says the force of a spring is proportional to its extension:

$$F_{ij} = -c_{ij}x_{ij} \tag{16.21}$$

The vector component x_{ij} is directed from the center of logic cell i to the center of logic cell j. The vector magnitude is calculated as either the Euclidean or Manhattan distance between the logic cell centers. The c_{ij} form the connectivity or cost matrix (the matrix element c_{ij} is the number of connections between logic cell i and logic cell j). If we want, we can also weight the c_{ij} to denote critical connections. Figure 16.27 illustrates the force-directed placement algorithm.

In the definition of connectivity (Section 15.7.1, "Measuring Connectivity") it was pointed out that the network graph does not accurately model connections for nets with more than two terminals. Nets such as clock nets, power nets, and global reset lines have a huge number of terminals. The force-directed placement algorithms usually make special allowances for these situations to prevent the largest

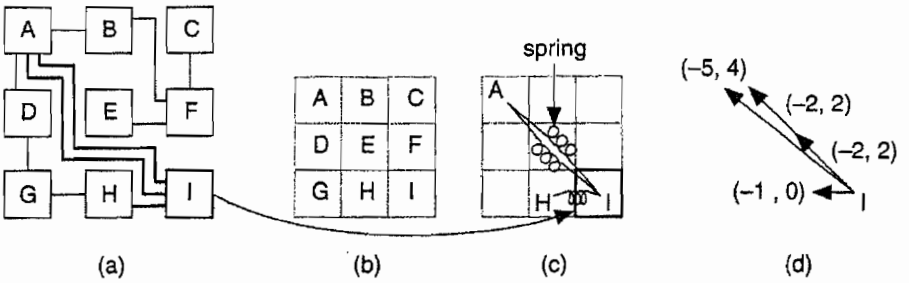


FIGURE 16.27 Force-directed placement. (a) A network with nine logic cells. (b) We make a grid (one logic cell per bin). (c) Forces are calculated as if springs were attached to the centers of each logic cell for each connection. The two nets connecting logic cells A and I correspond to two springs. (d) The forces are proportional to the spring extensions.

nets from snapping all the logic cells together. In fact, without external forces to counteract the pull of the springs between logic cells, the network will collapse to a single point as it settles. An important part of force-directed placement is fixing some of the logic cells in position. Normally ASIC designers use the I/O pads or other external connections to act as anchor points or fixed seeds.

Figure 16.28 illustrates the different kinds of force-directed placement algorithms. The **force-directed interchange** algorithm uses the force vector to select a

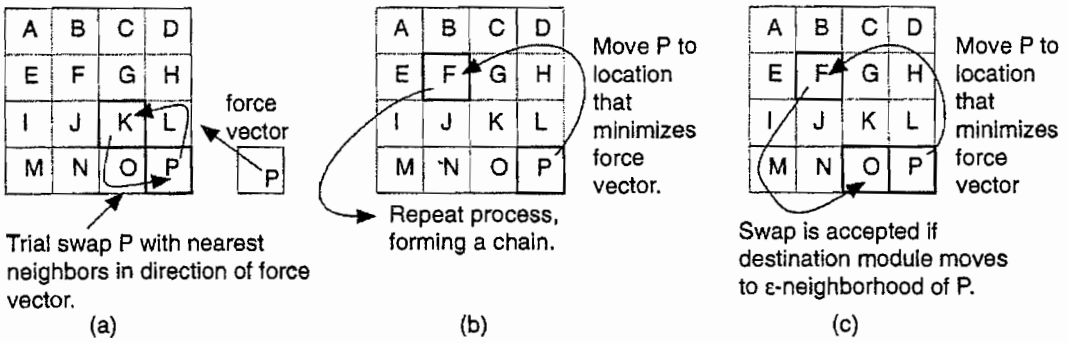


FIGURE 16.28 Force-directed iterative placement improvement. (a) Force-directed interchange. (b) Force-directed relaxation. (c) Force-directed pairwise relaxation.

pair of logic cells to swap. In **force-directed relaxation** a chain of logic cells is moved. The **force-directed pairwise relaxation** algorithm swaps one pair of logic cells at a time.

We reach a force-directed solution when we minimize the energy of the system, corresponding to minimizing the sum of the squares of the distances separating logic cells. Force-directed placement algorithms thus also use a quadratic cost function.

16.2.7 Placement Using Simulated Annealing

The principles of simulated annealing were explained in Section 15.7.8, "Simulated Annealing." Because simulated annealing requires so many iterations, it is critical that the placement objectives be easy and fast to calculate. The optimum connection pattern, the MRST, is difficult to calculate. Using the half-perimeter measure (Section 16.2.3) corresponds to minimizing the total interconnect length. Applying simulated annealing to placement, the algorithm is as follows:

1. Select logic cells for a trial interchange, usually at random.
2. Evaluate the objective function E for the new placement.
3. If ΔE is negative or zero, then exchange the logic cells. If ΔE is positive, then exchange the logic cells with a probability of $\exp(-\Delta E/T)$.
4. Go back to step 1 for a fixed number of times, and then lower the temperature T according to a cooling schedule: $T_{n+1} = 0.9\,T_n$, for example.

Kirkpatrick, Gerlatt, and Vecchi first described the use of simulated annealing applied to VLSI problems [1983]. Experience since that time has shown that simulated annealing normally requires the use of a slow cooling schedule and this means long CPU run times [Sechen, 1988; Wong, Leong, and Liu, 1988]. As a general rule, experiments show that simple min-cut based constructive placement is faster than simulated annealing but that simulated annealing is capable of giving better results at the expense of long computer run times. The iterative improvement methods that we described earlier are capable of giving results as good as simulated annealing, but they use more complex algorithms.

While I am making wild generalizations, I will digress to discuss **benchmarks** of placement algorithms (or any CAD algorithm that is random). It is important to remember that the results of random methods are themselves random. Suppose the results from two random algorithms, A and B, can each vary by ±10 percent for any chip placement, but both algorithms have the same average performance. If we compare single chip placements by both algorithms, they could falsely show algorithm A to be better than B by up to 20 percent or vice versa. Put another way, if we run enough test cases we will eventually find some for which A is better than B by 20 percent—a trick that Ph.D. students and marketing managers both know well. Even single-run evaluations over multiple chips is hardly a fair comparison. The only way to obtain meaningful results is to compare a statistically meaningful number of runs for a statistically meaningful number of chips for each algorithm. This same caution applies to any VLSI algorithm that is random. There was a Design Automation Conference panel session whose theme was "Enough of algorithms claiming improvements of 5 %."

16.2.8 Timing-Driven Placement Methods

Minimizing delay is becoming more and more important as a placement objective. There are two main approaches: net based and path based. We know that we can use net weights in our algorithms. The problem is to calculate the weights. One method finds the n most critical paths (using a timing-analysis engine, possibly in the synthesis tool). The net weights might then be the number of times each net appears in this list. The problem with this approach is that as soon as we fix (for example) the first 100 critical nets, suddenly another 200 become critical. This is rather like trying to put worms in a can—as soon as we open the lid to put one in, two more pop out.

Another method to find the net weights uses the **zero-slack algorithm** [Hauge et al., 1987]. Figure 16.29 shows how this works (all times are in nanoseconds). Figure 16.29(a) shows a circuit with **primary inputs** at which we know the **arrival times** (this is the original definition, some people use the term **actual times**) of each signal. We also know the **required times** for the **primary outputs**—the points in time at which we want the signals to be valid. We can work forward from the primary inputs and backward from the primary outputs to determine arrival and required times at each input pin for each net. The difference between the required and arrival times at each input pin is the **slack time** (the time we have to spare). The zero-slack algorithm adds delay to each net until the slacks are zero, as shown in Figure 16.29(b). The net delays can then be converted to weights or constraints in the placement. Notice that we have assumed that all the gates on a net switch at the same time so that the net delay can be placed at the output of the gate driving the net—a rather poor timing model but the best we can use without any routing information.

An important point to remember is that adjusting the net weight, even for every net on a chip, does not theoretically make the placement algorithms any more complex—we have to deal with the numbers anyway. It does not matter whether the net weight is 1 or 6.6, for example. The practical problem, however, is getting the weight information for each net (usually in the form of timing constraints) from a synthesis tool or timing verifier. These files can easily be hundreds of megabytes in size (see Section 16.4).

With the zero-slack algorithm we simplify but overconstrain the problem. For example, we might be able to do a better job by making some nets a little longer than the slack indicates if we can tighten up other nets. What we would really like to do is deal with *paths* such as the critical path shown in Figure 16.29(a) and not just *nets*. Path-based algorithms have been proposed to do this, but they are complex and not all commercial tools have this capability (see, for example, [Youssef, Lin, and Shragowitz, 1992]).

There is still the question of how to predict path delays between gates with only placement information. Usually we still do not compute a routing tree but use simple approximations to the total net length (such as the half-perimeter measure) and then use this to estimate a net delay (the same to each pin on a net). It is not until the routing step that we can make accurate estimates of the actual interconnect delays.

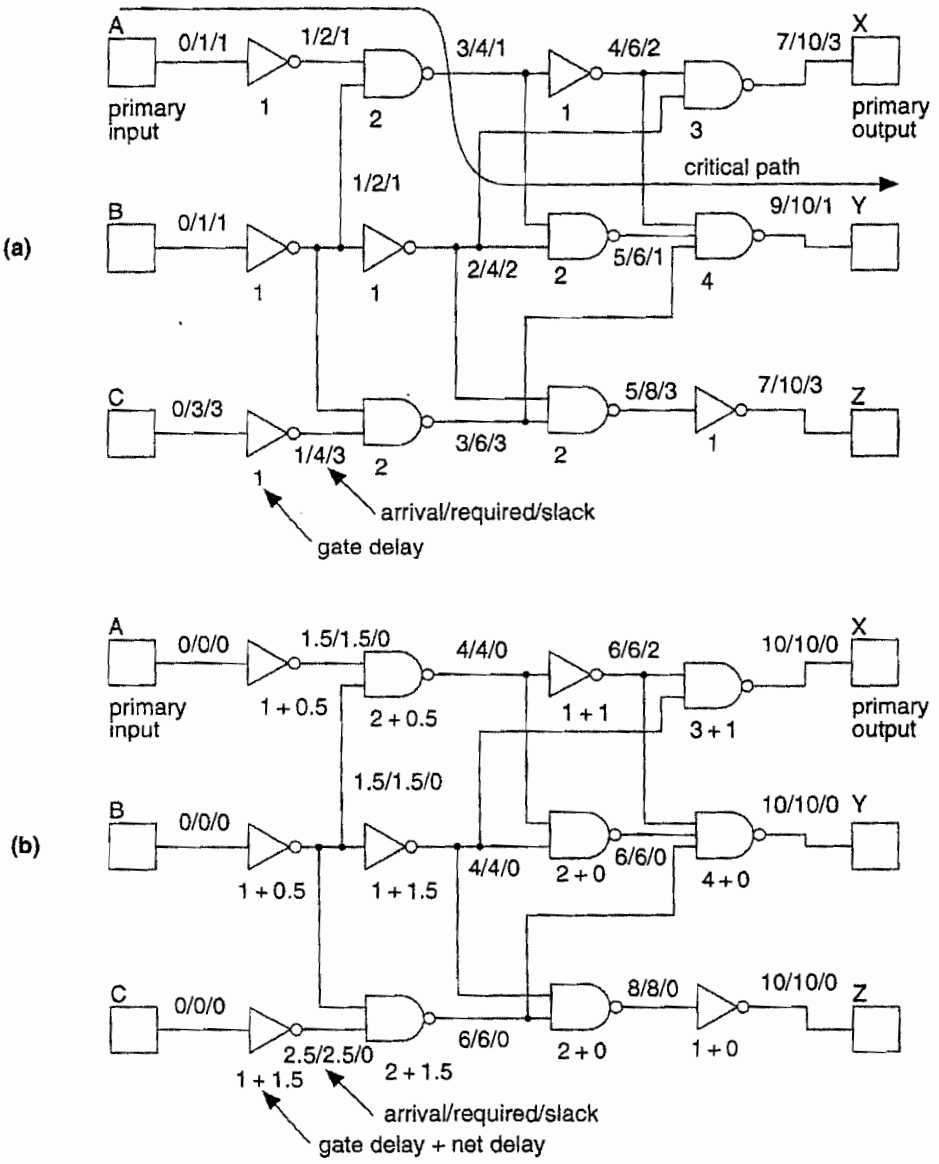


FIGURE 16.29 The zero-slack algorithm. (a) The circuit with no net delays. (b) The zero-slack algorithm adds net delays (at the outputs of each gate, equivalent to increasing the gate delay) to reduce the slack times to zero.

16.2.9 A Simple Placement Example

Figure 16.30 shows an example network and placements to illustrate the measures

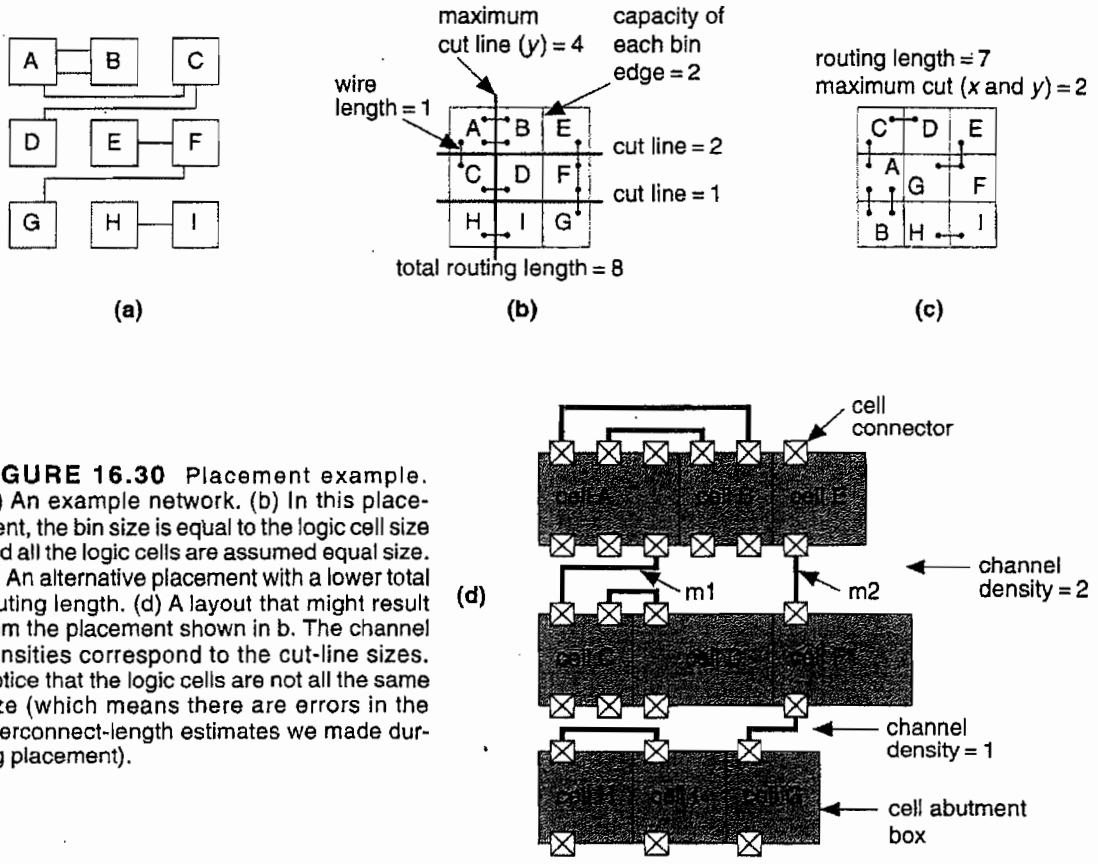


FIGURE 16.30 Placement example. (a) An example network. (b) In this placement, the bin size is equal to the logic cell size and all the logic cells are assumed equal size. (c) An alternative placement with a lower total routing length. (d) A layout that might result from the placement shown in b. The channel densities correspond to the cut-line sizes. Notice that the logic cells are not all the same size (which means there are errors in the interconnect-length estimates we made during placement).

for interconnect length and interconnect congestion. Figure 16.30(b) and (c) illustrate the meaning of total routing length, the maximum cut line in the x-direction, the maximum cut line in the y-direction, and the maximum density. In this example we have assumed that the logic cells are all the same size, connections can be made to terminals on any side, and the routing channels between each adjacent logic cell have a capacity of 2. Figure 16.30(d) shows what the completed layout might look like.

16.3 Physical Design Flow

Historically placement was included with routing as a single tool (the term P&R is often used for place and route). Because interconnect delay now dominates gate delay, the trend is to include placement within a floorplanning tool and use a separate router. Figure 16.31 shows a design flow using synthesis and a floorplanning tool that includes placement. This flow consists of the following steps:

1. *Design entry.* The input is a logical description with no physical information.
2. *Synthesis.* The initial synthesis contains little or no information on any interconnect loading. The output of the synthesis tool (typically an EDIF netlist) is the input to the floorplanner.
3. *Initial floorplan.* From the initial floorplan interblock capacitances are input to the synthesis tool as load constraints and intrablock capacitances are input as wire-load tables.
4. *Synthesis with load constraints.* At this point the synthesis tool is able to resynthesize the logic based on estimates of the interconnect capacitance each

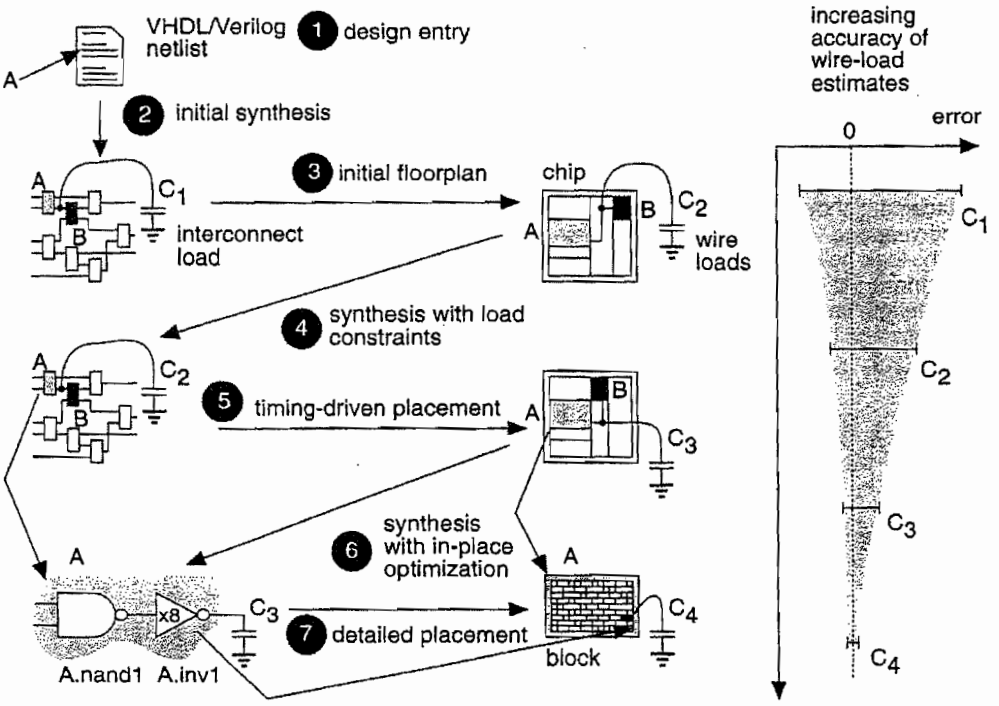


FIGURE 16.31 Timing-driven floorplanning and placement design flow. Compare with Figure 15.1 on p. 806.

gate is driving. The synthesis tool produces a forward annotation file to constrain path delays in the placement step.

5. *Timing-driven placement.* After placement using constraints from the synthesis tool, the location of every logic cell on the chip is fixed and accurate estimates of interconnect delay can be passed back to the synthesis tool.
6. *Synthesis* with **in-place optimization (IPO)**. The synthesis tool changes the drive strength of gates based on the accurate interconnect delay estimates from the floorplanner without altering the netlist structure.
7. *Detailed placement.* The placement information is ready to be input to the routing step.

In Figure 16.31 we iterate between floorplanning and synthesis, continuously improving our estimate for the interconnect delay as we do so.

16.4 Information Formats

With the increasing importance of interconnect a great deal of information needs to flow between design tools. There are some de facto standards that we shall look at next. Some of the companies involved are working toward releasing these formats as IEEE standards.

16.4.1 SDF for Floorplanning and Placement

In Section 13.5.6, "SDF in Simulation," we discussed the structure and use of the standard delay format (SDF) to describe gate delay and interconnect delay. We may also use SDF with floorplanning and synthesis tools to **back-annotate** an interconnect delay. A synthesis tool can use this information to improve the logic structure. Here is a fragment of SDF:

```
(INSTANCE B) (DELAY (ABSOLUTE
  (INTERCONNECT A.INV8.OUT B.DFF1.Q (:0.6:) (:0.6:))))
```

In this example the rising and falling delay is 60 ps (equal to 0.6 units multiplied by the time scale of 100 ps per unit specified in a `TIMESCALE` construct that is not shown). The delay is specified between the output port of an inverter with instance name `A.INV8` in block A and the Q input port of a D flip-flop (instance name `B.DFF1`) in block B. A `'.'` (period or fullstop) is set to be the hierarchy divider in another construct that is not shown.

There is another way of specifying interconnect delay using `NETDELAY` (a short form of the `INTERCONNECT` construct) as follows:

```
(TIMESCALE 100ps) (INSTANCE B) (DELAY (ABSOLUTE
(NETDELAY net1 (0.6)))
```

In this case all delays from an output port to, possibly multiple, input ports have the same value (we can also specify the output port name instead of the net name to identify the net). Alternatively we can lump interconnect delay at an input port:

```
(TIMESCALE 100ps) (INSTANCE B.DFF1) (DELAY (ABSOLUTE
  (PORT CLR (16:18:22) (17:20:25))))
```

This PORT construct specifies an interconnect delay placed at the input port of a logic cell (in this case the CLR pin of a flip-flop). We do not need to specify the start of a path (as we do for INTERCONNECT).

We can also use SDF to **forward-annotate** path delays using **timing constraints** (there may be hundreds or thousands of these in a file). A synthesis tool can pass this information to the floorplanning and placement steps to allow them to create better layout. SDF describes timing checks using a range of TIMINGCHECK constructs. Here is an example of a single path constraint:

```
(TIMESCALE 100ps) (INSTANCE B) (TIMINGCHECK
  (PATHCONSTRAINT A.AOI22_1.O B.ND02_34.O (0.8) (0.8)))
```

This describes a constraint (keyword PATHCONSTRAINT) for the rising and falling delays between two ports at each end of a path (which may consist of several nets) to be less than 80 ps. Using the SUM construct we can constrain the sum of path delays to be less than a specific value as follows:

```
(TIMESCALE 100ps) (INSTANCE B) (TIMINGCHECK
  (SUM (AOI22_1.O ND02_34.I1) (ND02_34.O ND02_35.I1) (0.8)))
```

We can also constrain skew between two paths (in this case to be less than 10 ps) using the DIFF construct:

```
(TIMESCALE 100ps) (INSTANCE B) (TIMINGCHECK
  (DIFF (A.I_1.O B.ND02_1.I1) (A.I_1.O.O B.ND02_2.I1) (0.1)))
```

In addition we can constrain the skew between a reference signal (normally the clock) and all other ports in an instance (again in this case to be less than 10 ps) using the SKEWCONSTRAINT construct:

```
(TIMESCALE 100ps) (INSTANCE B) (TIMINGCHECK
  (SKEWCONSTRAINT (posedge clk) (0.1)))
```

At present there is no easy way in SDF to constrain the skew between a reference signal and other signals to be greater than a specified amount.

16.4.2 PDEF

The **physical design exchange format** (**PDEF**) is a proprietary file format used by Synopsys to describe placement information and the clustering of logic cells. Here is a simple, but complete PDEF file:

```
(CLUSTERFILE
  (PDEFVERSION "1.0")
```

```
(DESIGN "myDesign")
(DATE "THU AUG 6 12:00 1995")
(VENDOR "ASICS_R_US")
(PROGRAM "PDEF_GEN")
(VERSION "V2.2")
(DIVIDER .)
(CLUSTER (NAME "ROOT")
  (WIRE_LOAD "10mm x 10mm")
  (UTILIZATION 50.0)
  (MAX_UTILIZATION 60.0)
  (X_BOUNDS 100 1000)
  (Y_BOUNDS 100 1000)
    (CLUSTER (NAME "LEAF_1")
      (WIRE_LOAD "50k gates")
      (UTILIZATION 50.0)
      (MAX_UTILIZATION 60.0)
      (X_BOUNDS 100 500)
      (Y_BOUNDS 100 200)
      (CELL (NAME L1.RAM01)
      (CELL (NAME L1.ALU01)
      )
  )
)
```

This file describes two clusters:

- `ROOT`, which is the top-level (the whole chip). The file describes the size (*x*- and *y*-bounds), current and maximum area utilization (i.e., leaving space for interconnect), and the name of the wire-load table, `'10mm x 10mm'`, to use for this block, chosen because the chip is expected to be about 10 mm on a side.
- `LEAF_1`, a block below the top level in the hierarchy. This block is to use predicted capacitances from a wire-load table named `'50k gates'` (chosen because we know there are roughly 50 k-gate in this block). The `LEAF_1` block contains two logic cells: `L1.RAM01` and `L1.ALU01`.

16.4.3 LEF and DEF

The **library exchange format** (**LEF**) and **design exchange format** (**DEF**) are both proprietary formats originated by Tangent in the TanCell and TanGate place-and-route tools which were bought by Cadence and now known as Cell3 Ensemble and Gate Ensemble respectively. These tools, and their derivatives, are so widely used that these formats have become a de facto standard. LEF is used to define an IC process and a logic cell library. For example, you would use LEF to describe a gate array: the base cells, the legal sites for base cells, the logic macros with their size and connectivity information, the interconnect layers and other information to set up the database that the physical design tools need. You would use DEF to describe all the physical aspects of a particular chip design including the netlist and physical location

of cells on the chip. For example, if you had a complete placement from a floorplanning tool and wanted to exchange this information with Cadence Gate Ensemble or Cell3 Ensemble, you would use DEF.

16.5 Summary

Floorplanning follows the system partitioning step and is the first step in arranging circuit blocks on an ASIC. There are many factors to be considered during floorplanning: minimizing connection length and signal delay between blocks; arranging fixed blocks and reshaping flexible blocks to occupy the minimum die area; organizing the interconnect areas between blocks; planning the power, clock, and I/O distribution. The handling of some of these factors may be automated using CAD tools, but many still need to be dealt with by hand. Placement follows the floorplanning step and is more automated. It consists of organizing an array of logic cells within a flexible block. The criterion for optimization may be minimum interconnect area, minimum total interconnect length, or performance. There are two main types of placement algorithms: based on min-cut or eigenvector methods. Because interconnect delay in a submicron CMOS process dominates logic-cell delay, planning of interconnect will become more and more important. Instead of completing synthesis before starting floorplanning and placement, we will have to use synthesis and floorplanning/placement tools together to achieve an accurate estimate of timing.

The key points of this chapter are:

- Interconnect delay now dominates gate delay.
- Floorplanning is a mapping between logical and physical design.
- Floorplanning is the center of ASIC design operations for all types of ASIC.
- Timing-driven floorplanning is becoming an essential ASIC design tool.
- Placement is now an automated function.

16.6 Problems

* = Difficult, ** = Very difficult, *** = Extremely difficult

16.1 (Wire loads, 30 min.) Table 16.2 shows a wire-load table. Since you might expect the interconnect load to be a monotonic increasing function of fanout and block area, it seems as though some of the data in Table 16.2 may be in error; these figures are shown preceded by an asterisk, '*' (this table is from an ASIC vendor data book). Using a spreadsheet, analyze the data in Table 16.2.

a. By graphing the data, indicate any figures in Table 16.2 that you think might be in error. If you think that there is an error, predict the correct values—either by interpolation (for values in error in the body of the table), or by fit-

ting the linear model parameters, the slope and the intercept (for any values in error in the last two columns of the table).

b. Including any corrections, how accurate is the model that predicts load as a linear function of fanout for a given block size? (Use the maximum error of the linear model expressed as a percentage of the table value.)

c. Can you fit a simple function to the (possibly corrected) figures in the last column of the table and explain its form?

d. What did you learn about wire-load tables from this problem?

TABLE 16.2 Wire-load table. Predicted interconnect loads (measured in standard loads) as a function of block size and fanout (Problem 16.1).

Size ($/mm^2$)	Fanout											Slope	Intercept
	1	2	3	4	5	6	7	8	16	32	64		
0.5×0.5	0.65	0.95	1.25	1.54	1.84	2.14	2.44	2.74	5.13	9.91	19.47	0.299	0.349
1×1	0.80	1.20	1.59	1.99	2.39	2.79	3.19	3.59	6.77	13.15	25.9	0.398	0.398
2×2	0.96	1.48	1.99	2.51	3.02	3.54	4.05	4.57	8.68	16.92	33.38	0.515	0.448
3×3	1.20	1.83	2.46	3.09	3.72	4.35	4.98	5.61	10.66	20.75	40.94	0.631	0.564
4×4	1.41	2.11	2.81	3.50	4.20	4.90	5.59	6.29	11.87	23.02	45.33	0.697	0.714
5×5	1.51	2.24	2.97	3.70	4.43	5.16	5.89	6.62	12.47	24.15	47.53	0.730	0.780
6×6	1.56	2.31	3.05	3.80	4.55	5.30	6.04	6.79	12.77	24.72	48.62	0.747	0.813
7×7	1.83	2.62	3.42	4.22	5.01	5.81	6.61	7.40	13.78	26.53	52.02	0.797	*1.029
8×8	1.88	2.74	3.6	4.47	5.33	6.19	7.06	7.92	14.82	26.64	56.26	0.863	1.013
9×9	2.01	2.94	3.87	4.80	5.73	6.66	7.59	8.52	15.95	30.83	60.57	0.930	1.079
10×10	2.01	2.98	3.94	4.90	5.86	6.83	7.79	8.75	16.45	31.86	62.67	0.963	*1.050
11×11	2.46	3.46	4.45	5.45	6.44	7.44	8.44	9.43	17.4	33.33	65.20	0.996	1.465
12×12	3.04	4.1	5.17	6.23	7.3	8.35	9.42	10.48	18.8	36.03	70.00	1.063	1.964

16.2 (Trees, 20 min.) For the network graph shown in Figure 16.32(f), draw the following trees and calculate their Manhattan lengths:

a. The minimum Steiner tree.

b. The **chain connection**.

c. The minimum rectilinear Steiner tree.

d. The **minimum rectilinear spanning tree** [Hwang, 1976].

e. The minimum single-trunk rectilinear Steiner tree (with a horizontal or vertical trunk).

f. The **minimum rectilinear chain connection** (easy to compute).

g. The **minimum source-to-sink connection.**

Calculate:

h. The complete-graph measure and the half-perimeter measure.

Figure 16.32 parts (a–e) illustrate the definitions of these trees. There is no known solution to the minimum Steiner-tree problem for nets with more than five terminals.

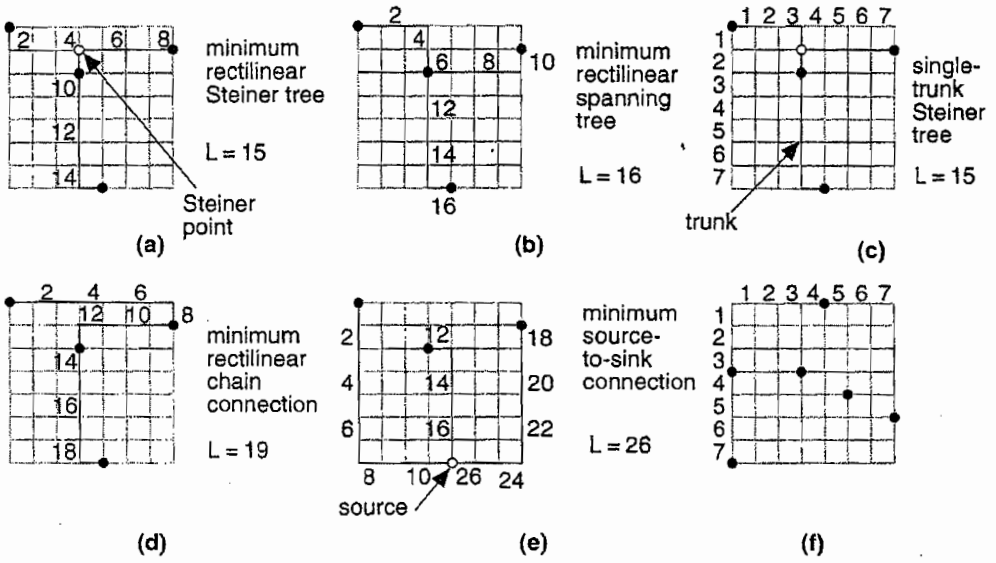


FIGURE 16.32 Tree routing. (a) The minimum rectilinear Steiner tree (MRST). (b) The minimum rectilinear spanning tree. (c) The minimum single-trunk rectilinear Steiner tree (1-MRST). (d) The minimum rectilinear chain connection. (e) The minimum source-to-sink connection. (f) Example net for Problem 16.2.

16.3 (Eigenvalue placement constraints, 10 min. [Cheng and Kuh, 1984]) Consider the one-dimensional placement problem with a vector list of valid positions for the logic cells $\mathbf{p} = [p_i]$ and a vector list of x-coordinates for the logic cells $\mathbf{x} = [x_i]$.

Show that for a valid placement $\mathbf{x}$ (where the vector elements x_i are some permutation of the vector elements p_i), the following equations hold:

$$\sum_{i=1}^{n} x_i = \sum_{i=1}^{n} p_i \qquad \sum_{i=1}^{n} x_i^2 = \sum_{i=1}^{n} p_i^2 \qquad \cdots \qquad \sum_{i=1}^{n} x_i^n = \sum_{i=1}^{n} p_i^n \qquad (16.22)$$

(*Hint:* Consider the polynomial $(x + x_i)^n$. In our simplification to the problem, we chose to impose only the second equation of these constraints.)

16.4 (*Eigenvalue placement, 30 min.) You will need MatLab, Mathematica, or a similar mathematical calculus program for this problem.

a. Find the eigenvalues and eigenvectors for the disconnection matrix corresponding to the following connection matrix:

```
C=
[ 0 1 1 0 0 0 1 0 0;
  1 0 0 0 0 0 0 0 0;
  1 0 0 1 0 0 0 1 0;
  0 0 1 0 0 1 0 0 0;
  0 0 0 0 0 1 0 0 1;
  0 0 0 1 1 0 1 0 0;
  1 0 0 0 0 1 0 0 0;
  0 0 1 0 0 0 0 0 1;
  0 0 0 0 1 0 0 1 0;]
```

(*Hint:* Check your answer. The smallest, nonzero, eigenvalue should be 0.5045.)

b. Use your results to place the logic cells. Plot the placement and show the connections between logic cells (this is easy to do using an X-Y plot in an Excel spreadsheet).

c. Check that the following equation holds:

$$\lambda = \frac{g}{P}.$$

16.5 (Die size, 10 min.) Suppose the minimum spacing between pad centers is W mil (1 mil = 10^{-3} inch), there are N I/O pads on a chip, and the die area (assume a square die) is A mil^2:

a. Derive a relationship between W, N, and A that corresponds to the point at which the die changes from being pad-limited to core-limited.

b. Plot this relationship with N (ranging from 50 to 500 pads) on the x-axis, A on the y-axis (for dies ranging in size from 1 mm to 20 mm on a side), and W as a parameter (for $W = 1$, 2, 3, and 4 mil).

16.6 (Power buses, 20 min.) Assume aluminum metal interconnect has a resistance of about 30 mΩ/square (a low value). Consider a power ring for the I/O pads. Suppose you have a high-power chip that dissipates 5 W at $V_{DD} = 5$ V, and assume that half of the supply current (0.5 A) is due to I/O. Suppose the square die is L mil on a side, and that the I/O current is equally distributed among the N VDD pads that are on the chip. In the worst case, you want no more than 100 mV drop between any VDD pad and the I/O circuits drawing power (notice that there will be an equal drop on the VSS side; just consider the VDD drop).

a. Model the power distribution as a ring of N equally spaced pads. Each pad is connected by a resistor equal to the aluminum VDD power-bus resistance between two pads. Assume the I/O circuits associated with each pad can be

considered to connect to just one point on the resistors between each pad. If the resistance between each pad is R, what is the worst-case resistance between the I/O circuits and the supply?

b. Plot a graph showing L (in mil) on the x-axis, W (the required power-bus width in microns) on the y-axis, with N as a parameter (with $N = 1, 2, 5, 10$).

c. Comment on your results.

d. An upper limit on current density for aluminum metallization is about 50 $kAcm^{-2}$; at current densities higher than this, failure due to electromigration (which we shall cover in Section 17.3.2, "Power Routing") is a problem. Assume the metallization is 0.5 μm thick. Calculate the current density in the VDD power bus for this chip in terms of the power-bus width and the number of pads. Comment on your answer.

16.7 (Interconnect-length approximation, 10 min.) Figure 16.22 shows the correlation between actual interconnect length and two approximations. Use this graph to derive a correction function (together with an estimation of the error) for the complete-graph measure and the half-perimeter measure.

16.8 (Half-perimeter measure, 10 min.) Draw a tree on a rectangular grid for which the MRST is equal to the half-perimeter measure. Draw a tree on a rectangular grid for which the MRST is twice the half-perimeter measure.

16.9 (***Min-cut, 120 min.) Many floorplanning and placement tools use min-cut methods and allow you to alter the type and sequence of bisection cuts. Research and describe the difference between: quadrature min-cut placement, bisection min-cut placement, and slice/bisection min-cut placement.

16.10 (***Terminal propagation, 120 min.) There is a problem with the min-cut algorithm in the way connectivity is measured. Figure 16.33 shows a situation in which logic cells G and H are connected to other logic cells (A and F) outside the area R_1 that is currently being partitioned. The min-cut algorithm ignores connections outside the area to be divided. Thus logic cells G and H may be placed in partition R_3 rather than partition R_2. Suggest solutions to this problem. *Hint:* See Dunlop [1983]; Hartoog [1986]; or the Barnes–Hut galaxy model.

16.11 (Benchmarks and statistics, 30 min.) Your boss asks you to compare two placement programs from companies ABC and XYZ. You run five test cases for both on a single netlist, P1. You get results (measured in arbitrary units) of 9, 8, 9, 7, 11 for ABC; 6, 9, 10, 13, 8 for XYZ.

a. Calculate the mean and standard deviations for these results.

b. What confidence (in the statistical sense) do you have in these figures?

c. What can you say about the relative performance of ABC and XYZ?

On average each test case takes about 0.5 hours (wall clock) for both ABC and XYZ. Next you run six test cases on another netlist, P2 with the following results: 4, 6, 7, 8, 5, 7 for ABC, and 4, 5, 3, 6, 4, 3 for XYZ. These test cases take about 0.75 hours (wall clock) each.

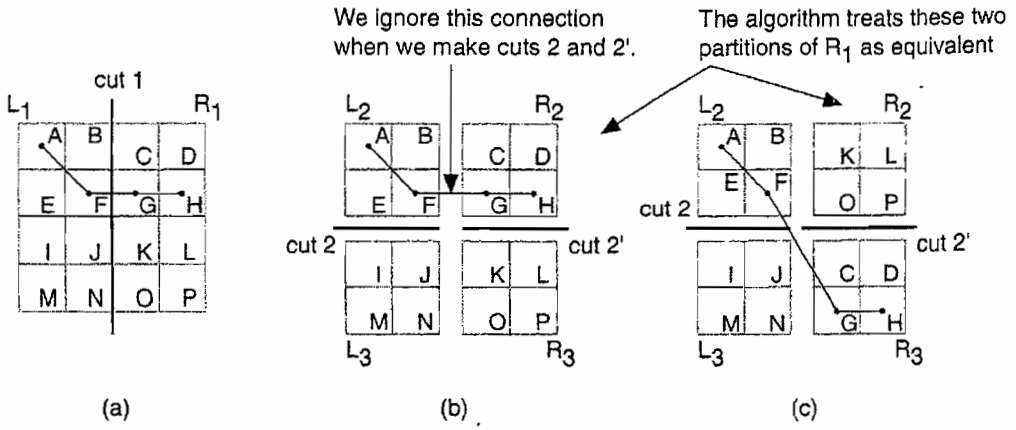


FIGURE 16.33 (For Problem 16.10.) A problem with the min-cut algorithm is that it ignores connections to logic cells outside the area being partitioned. (a) We perform a vertical cut 1 producing the areas R_1 and R_2. (b) Next we make a horizontal cut 2, producing L_2 and L_3, and a cut 2', producing R_2 and R_3. (c) The min-cut algorithm ignores the connection from L_2 and is equally likely to produce the arrangement shown here when we make cut 2'.

d. What can you say about the P2 results?

e. Given the P1 and P2 results together, what can you say about ABC and XYZ?

f. How many P1 test cases should you run to get a result so that you can say ABC is better or worse than XYZ with 90 percent confidence (i.e., you make the right decision 9 out of 10 times)? How long would this take?

g. Find the same figures for the P2 netlist. Comment on your answers.

h. Suppose you had more netlists and information about the variation of results from each netlist, together with the average time to run each netlist. How would you use this information to get the most meaningful result in the shortest time?

16.12 (Linear and quadratic placement, 20 min.) [Sigl, Doll, and Johannes, 1991] Figure 16.34(a) shows a simple network that we will place. Figure 16.34(b) shows the problem. The logic cells are all the same size: 1 grid unit wide by 1 grid unit high. Logic cells 1 and 3 are fixed at the locations shown. Logic cell 2 is movable and placed at coordinates (for the lower-left corner) of (x_2, y_2). The lower-left corners of logic cells should be placed at grid locations and should not overlap.

a. What is the connection matrix c_{ij} for this network?

b. Calculate and draw (showing the logic-cell coordinates) the placement that minimizes the linear cost function (or objective function) f_L,

$$f_L = \frac{1}{2} \sum_{i,j=1}^{n} c_{ij} d_{ij} \tag{16.23}$$

where d_{ij} is the distance between logic cells i and j.

c. Calculate and draw (showing coordinates) the placement that minimizes the quadratic cost function f_Q,

$$f_Q = \frac{1}{2} \sum_{i,j=1}^{n} c_{ij} d_{ij}^2 . \tag{16.24}$$

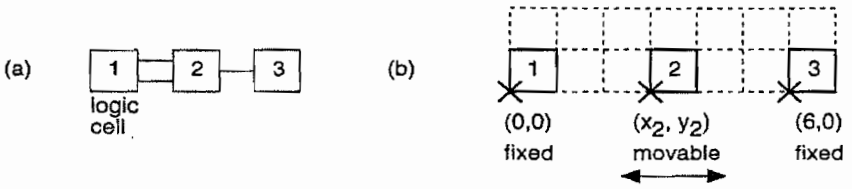


FIGURE 16.34 Problem 16.12 illustrates placement objectives. (a) An example network for placement. (b) The placement restrictions. Logic cells 1 and 3 are fixed in position, the placement problem is to optimize the position of logic cell 2 under different placement objectives.

16.13 (Placement interconnect lengths, 45 min.) Figure 16.30(d) shows the actual routing corresponding to a placement with an estimated routing length of 8 units (Figure 16.30b).

a. Draw the layout (with routing) corresponding to the placement of Figure 16.30(c), which has a lower estimated total routing length of 7 units.

b. Compare the actual total routing length for both layouts and explain why they are different from the estimated lengths and describe the sources of the errors.

c. Consider flipping both logic cells A and B about the y-axis in the layout shown in Figure 16.30(d). How much does this shorten the total interconnect length? Some placement algorithms consider such moves.

16.14 (Zero-slack algorithm, 60 min.) For the circuit of Figure 16.35:

a. Find all of the arrival, required, and slack times (all delays are in nanoseconds).

b. What is the critical path?

c. If the gate delay of A2 is increased to 5 ns, what is the new critical path?

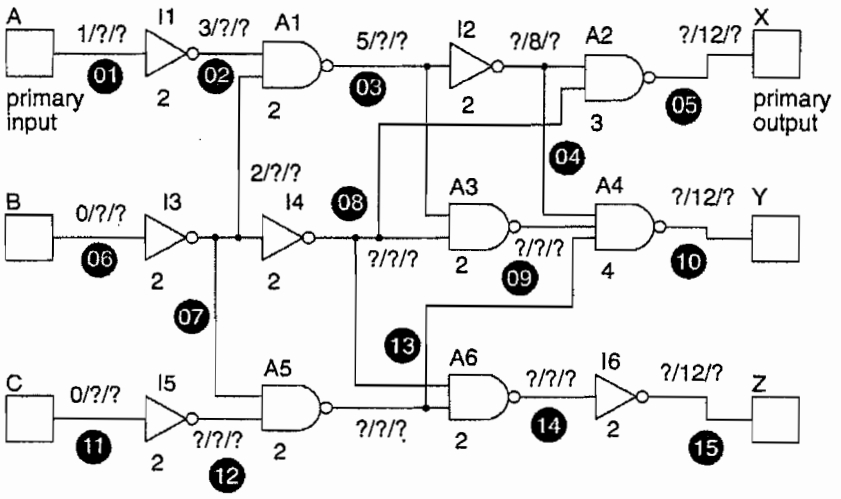


FIGURE 16.35 A circuit to illustrate the zero-slack algorithm (Problem 16.14).

d. ** Using your answer to part a find the upper bounds on net delays by means of the zero-slack algorithm as follows:

i. Find arrival, required, and slack times on all nets.
ii. Find an input pin p with the least nonzero slack S_p on a net which has not already been selected. If there are none go to step 6.
ii. Find the path through p (may include several gates) on which all pins have slack S_p.
iv. Distribute a delay equal to the slack S_p along the path assigning a fraction to each net at the output pins of the gates on the path.
v. Work backward from p updating all the required times as necessary and forward from p updating all the arrival times.
vi. Convert net delays to net lengths.

Hint: You can consult the original description of the zero-slack algorithm if this is not clear [Hauge et al., 1987].

16.15 (World planning, 60 min.) The seven continents are (with areas in millions of square miles): Europe—strictly a peninsula of Asia (4.1), Asia (17.2), North America (9.4), South America (6.9), Australia (3.0), Africa (11.7), and Antarctica (5.1). Assume the continents are flexible blocks whose aspect ratio may be adjusted.

a. Create a slicing floorplan of the world with a square aspect ratio.

b. Draw a world connectivity graph with seven nodes and whose edges are labeled with the distances between Moscow, Beijing, Chicago, Rio de Janeiro, Sydney, Nairobi, and the South Pole.

c. Suppose you want to floorplan the world so that the difference in distances between the centers of the continental blocks and the corresponding edges in the world connectivity graph is minimized. How would you measure the differences in distance? Suggest a method to minimize your measure.

d. Use an eigenvalue method to floorplan the world. Draw the result with coordinates for each block and explain your approach.

16.7 Bibliography

There are no recent monographs or review articles on floorplanning modern ASICs with interconnect delay dominating gate delay.,Placement is a much more developed topic. Perhaps the simplest place to dig deeper is the book by Preas and Lorenzetti that contains a chapter titled "Placement, assignment, and floorplanning" [Preas and Karger, 1988]. The collection edited by Ohtsuki [1986] contains a review paper by Yoshida titled "Partitioning, assignment, and placement." Sangiovanni-Vincentelli's review article [1986] complements Ohtsuki's edited book, but both are now dated. Sechen's book [1988] describes simulated annealing and its application to placement and chip-planning for standard cell and gate array ASICs. Part III of the IEEE Press book edited by Hu and Kuh [1983] is a collection of papers on wireability, partitioning, and placement covering some of the earlier and fundamental work in this area. For a more recent and detailed look at the inner workings of floorplanning and placement tools, Lengauer's [1990] book on algorithms contains a chapter on graph algorithms and a chapter on placement, assignment, and floorplanning. Most of these earlier book references deal with placement before the use of timing as an additional objective. The tutorial paper by Benkoski and Strojwas [1991] contains a number of references on performance-driven placement. Luk's book [1991] describes methods for estimating net delay during placement.

Papers and tutorials on all aspects of floorplanning and placement (with an emphasis on algorithms) are published in *IEEE Transactions on Computer-Aided Design*. The newest developments in floorplanning and placement appear every year in the *Proceedings of the ACM/IEEE Design Automation Conference* (DAC) and *Proceedings of the IEEE International Conference on Computer-Aided Design* (ICCAD).

16.8 References

Page numbers in brackets after a reference indicate its location in the chapter body.

Benkoski, J., and A. J. Strojwas. 1991. "The role of timing verification in layout synthesis." In *Proceedings of the 28th ACM/IEEE Design Automation Conference*, San Francisco, pp. 612–619. Tutorial paper with 60 references. This was an introduction to a session on Placement for Performance Optimization containing five other papers on this topic. [p. 906]

Breuer, M. A. 1977. "Min-cut placement." *Journal of Design Automation and Fault Tolerant Computing*, Vol. 1, no. 4, pp. 343–362. [p. 882]

Chao, A. H., E. M. Nequist, and T. D. Vuong. 1990. "Direct solution of performance constraints during placement." In *Proceedings of the IEEE Custom Integrated Circuits Conference*. Describes algorithms used in Cadence Gate Ensemble for performance-driven placement. Wiring estimate is based on single trunk Steiner tree with corrections for bounding rectangle aspect ratio and pin count. [p. 879]

Cheng, C.-K., and E. S. Kuh. 1984. "Module placement based on resistive network optimization." *IEEE Transactions on Computer-Aided Design for Integrated-Circuits and Systems*, Vol. CAD-3, pp. 218–225. [pp. 884, 900]

Dunlop, A. E., and B. W. Kernighan. 1983. "A placement procedure for polycell VLSI circuits." In *Proceedings of the IEEE International Conference on Computer Aided Design*, Santa Clara, CA, September 13–15. Describes the terminal propagation algorithm. [p. 902]

Goto, S. and T. Matsuda. 1986. "Partitioning, assignment and placement." In *Layout Design and Verification*, T. Ohtsuki (Ed.), Vol. 4, pp. 55–97. New York: Elsevier. ISBN 0444878947. TK 7874. L318. [p. 879]

Hall, K. M. 1970. "An r-dimensional quadratic placement algorithm." *Management Science*, Vol. 17, no. 3, pp. 219–229. [p. 885]

Hanan, M. 1966. "On Steiner's problem with rectilinear distance." *Journal SIAM Applied Mathematics*, Vol. 14, no. 2, pp. 255–265. [p. 879]

Hanan, M., P. K. Wolff Sr., and B. J. Agule. 1973. "Some experimental results on placement techniques." In *Proceedings of the 13th Design Automation Conference*. Reference to complete graph wire measure. [p. 877]

Hartoog, M. R., 1986. "Analysis of placement procedures for VLSI standard cell layout." In *Proceedings of the 23rd Design Automation Conference*. [p. 902]

Hauge, P. S., et al. 1987. "Circuit placement for predictable performance." In *Proceedings of the IEEE International Conference on Computer Aided Design*, pp. 88–91. Describes the zero-slack algorithm. *See also:* Nair, R., C. L. Berman, P. S. Hauge, and E. J. Yoffa, "Generation of performance constraints for layout," *IEEE Transactions on Computer Aided Design*, Vol. 8, no. 8, pp. 860–874, August 1989; and Burstein, M. and M. N. Housewife, "Timing influenced layout design," in *Proceedings of the 22nd Design Automation Conference*, 1985. Defines required, actual, and slack times. Describes application of timing-driven restrictions to placement using F–M algorithm and hierarchical global routing. [p. 905]

Hu, T. C., and E. S. Kuh (Eds.). 1983. *VLSI Circuit Layout: Theory and Design*. New York: IEEE Press. Contains 26 papers divided into six parts; Part 1: Overview; Part II: General; Part III: Wireability, Partitioning and Placement; Part IV: Routing; Part V: Layout Systems; Part VI: Module Generation. ISBN 0879421932. TK7874. V5573. [p. 906]

Hwang, F. K. 1976. "On Steiner minimal trees with rectilinear distance." *SIAM Journal of Applied Mathematics*, Vol. 30, pp. 104–114. *See also:* Hwang, F. K., "An O(n log n) Algorithm for Suboptimal Rectilinear Steiner Trees," *IEEE Transactions on Circuits and Systems*, Vol. CAS-26, no. 1, pp. 75–77, January 1979. Describes an algorithm to improve the rectilinear minimum spanning tree (RMST) approximation to the minimal rectilinear Steiner tree (minimal RST). The approximation is at most 1.5 times longer than the minimal RST, since the RMST is at worst 1.5 times the length of the minimal RST. [p. 899]

Kirkpatrick, S., C. D. Gerlatt Jr., and M. P. Vecchi. 1983. "Optimization by simulated annealing," *Science*, Vol. 220, no. 4598, pp. 671–680. [p. 890]

Lengauer, T. 1990. *Combinatorial Algorithms for Integrated Circuit Layout*. Chichester, England: Wiley. ISBN 0-471-92838-0. TK7874.L36. Contains chapters on circuit layout; optimization problems; graph algorithms; operations research and statistics; combinatorial

layout problems; circuit partitioning; placement; assignment; floorplanning; global routing and area routing; detailed routing; and compaction. 484 references. [p. 906]

Luk, W. K. 1991. "A fast physical constraint generator for timing driven layout." In *Proceedings of the 28th ACM/IEEE Design Automation Conference.* Introduction to timing-driven placement and net- and path-based approaches. Describes some different methods to estimate interconnect delay during placement. ISBN 0-89791-395-7. [p. 906].

Masleid, R. P. 1991. "High-density central I/O circuits for CMOS." *IEEE Journal of Solid-State Circuits,* Vol. 26, no. 3, pp. 431–435. An I/O circuit design that reduces the percentage of chip area occupied by I/O circuits from roughly 22 percent to under 3 percent for a 256 I/O chip. Uses IBM C4 technology that allows package connections to be located over chip circuitry. 10 references. [p. 866]

Ohtsuki, T. (Ed.). 1986. *Layout Design and Verification.* New York: Elsevier. Includes nine papers on CAD tools and algorithms: "Layout strategy, standardisation, and CAD tools," Ueda, Kasai and Sudo; "Layout compaction," Mylynski and Sung; "Layout verification," Yoshida; "Partitioning, assignment and placement," Goto and Matsuda; "Computational complexity of layout problems," Shing and Hu; "Computational and geometry algorithms," Asano, Sato and Ohtsuki; an excellent survey and tutorial paper by M. Burstein: "Channel routing"; "Maze-running and line-search algorithms" an easily-readable paper on detailed routing by Ohtsuki; and a mathematical paper, "Global routing," by Kuh and Marek-Sadowska. ISBN 0444878947. TK7874. L318. [p. 906]

Preas, B. T., and P. G. Karger. 1988. "Placement, assignment and floorplanning." In *Physical Design Automation of VLSI Systems,* B. T. Preas and M. J. Lorenzetti (Eds.), pp. 87–155. Menlo Park, CA: Benjamin-Cummings. ISBN 0-8053-0412-9. TK7874.P47. [p. 906]

Sangiovanni-Vincentelli, A. 1986. "Automatic layout of integrated circuits." In *Nato Advanced Study on "Logic Synthesis and Silicon Compilers for VLSI Design",* G. De Micheli, A. Sangiovanni-Vincentelli, and A. Paolo (Eds.). Norwell, MA: Kluwer. ISBN 90-247-2689-1, 90-247-3561-0. TK7874.N338. [p. 906]

Schweikert, D. G., 1976. "A 2-dimensional placement algorithm for the layout of electrical circuits." In *Proceedings of the 9th Design Automation Conference.* Description of half-perimeter wire measure. [p. 879]

Sechen, C. 1988. *VLSI Placement and Global Routing Using Simulated Annealing.* Norwell, MA: Kluwer. Contains chapters on the simulated annealing algorithm; placement and global routing; floorplanning; average interconnection length estimation; interconnect-area estimation; a channel definition algorithm; and a global router algorithm. ISBN 0898382815. TK7874. S38. [p. 890]

Sigl, G., K. Doll, and F. M. Johannes. 1991. "Analytical placement: a linear or quadratic objective function?" In *Proceedings of the 28th ACM/IEEE Design Automation Conference.* Compares quadratic and linear cost function for placement algorithms. Explains the Gordian place-and-route system from the Technical University of Munich. ISBN 0-89791-395-7. [p. 903].

Wada, T., M. Eino, and K. Anami. 1990. "Simple noise model and low-noise data-output buffer for ultrahigh-speed memories." *IEEE Journal of Solid-State Circuits,* Vol. 25, no. 6, pp. 1586–1588. An analytic noise model for voltage bounce on internal VDD/VSS lines. [p. 866]

Wong, D. F., H. W. Leong, and C. L. Liu. 1988. *Simulated Annealing for VLSI Design.* Norwell, MA: Kluwer. Introduction; Placement; Floorplan Design; Channel Routing; Permutation Channel Routing; PLA Folding; Gate Matrix Layout; Array Optimization. ISBN 0898382564. TK7874. W65. [p. 890]

Youssef, H., R.-B. Lin, and E. Shragowitz. 1992. "Bounds on net delays for VLSI circuits." *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing,* Vol. 39, no. 11, pp. 315–324. An alternative to the weight-based approach is development of delay bounds on all nets. 21 references. [p. 891].

TAB 34

PROCEEDINGS OF THE 33rd DESIGN AUTOMATION CONFERENCE

ACM Order Number 477960
ISBN 0-89791-779-0

Library of Congress Number 85-644924

IEEE Catalog Number 96CH35932
ISBN 0-7803-3364-0 (Softbound Edition)
ISBN 0-7803-3294-6 (Casebound Edition)
ISBN 0-7803-3295-4 (Microfiche Edition)
ISSN 0738-100X

Additional copies of 1996 or prior Proceedings may be ordered prepaid from

ACM Order Department
P.O. Box 12114
Church Street Station
New York, NY 10257

ACM European Service Center
108 Cowley Rd.
Oxford, OX41JF, U.K.
Phone: 44-1865-382338
Fax: 44-1865-381338
E-mail: ACM_Europe@acm.org

Phone: 1-800-342-6626
(U.S.A. and Canada)
+1-212-626-0500
(All other countries)
Fax: +1-212-944-1318
E-mail: acmhelp@acm.org

Additional copies of this publication are available from

IEEE Service Center
P.O. Box 1331
445 Hoes Lane
Piscataway, NJ 08855-1331

1-800-678-IEEE
1-908-981-1393
1-908-981-9667 (Fax)
833-233 (Telex)

Table of Contents

Session 4
Current Directions in High Level Synthesis

Chair: Hiroto Yasuura
Organizers: R. Walker, K. Wakabayashi

Session 5
Analysis and Synthesis of Asynchronous Circuits

Chair: Peter Beerel
Organizers: L. Lavagno, B. Lin

Session 6
New Frontiers in Partitioning

Chair: D. F. Wong
Organizers: A. Domic, A. B. Kahng

Session 7
Trends in Verification

Chair: James Rowson
Organizers: N. Collins, R. Goering

Panel: Hot New Trends in Verification

Session 8
Specialized Design Techniques for Speed and Power

Chair: Scott Nance
Organizers: D. Stark, B. Frye

Session 9
Test and Fault Tolerance in High Level Synthesis

Chair: C. Papachristou
Organizers: K. Wakabayashi, R. Camposano

Session 10
Issues in Discrete Simulation

Chair: Jay Lawrence
Organizers: R. McGeer, K. Sakallah

Session 11
Issues in Design Environments

Chair: Michaela Guiney
Organizers: D. Ku, R. A. Rutenbar

Session 12
Panel: Gearing Up for the Technology Explosion

Chair: Gary Smith
Organizer: M. Kenefick
Panelists: *Walt Davis, Glenn House, Kurt Keutzer, Jim Pena, Craig Peterson, Lawrence Rubin, Jim Solomon* 189

Session 13
Tutorial: The SPICE FET Models: Pitfalls and Prospects
(Are You An Educated Model Consumer?)

Chair: Daniel Foty
Organizer: J. Cooley
Presenter: *Daniel Foty* 190

Session 14
Combinational Logic Synthesis I

Chair: Gary D. Hachtel
Organizers: S. Malik, R. McGeer

Session 15
Pattern Generation for Test and Diagnosis

Chair: Janusz Rajski
Organizers: S. Kundu, Y. Zorian

Session 16
CAD for Analog and Mixed Signal ICs

Chair: James Spoto
Organizers: R. A. Rutenbar, J. White

Session 17
Panel: Core-Based Design for System-Level ASICs—Whose Job Is It?

Chair: Lynn Watson
Organizer: R. Goldman
Panelists: *Kim Asal, Andreas Danuser, Chris King, Susan Mason, Jim Pena, Scott Runner* 240

Session 18
Panel: A Common Standards Roadmap

Chair: Alain Hanover
Organizer: J. Smith
Panelists: *Rich Goldman, Andy Graham, Randolph E. Harr, Gregory W. Ledenbach, A. Richard Newton, Robert Rozeboom, Tabuchi Kinya* 241

Session 19
Combinational Logic Synthesis II

Chair: Iris Bahar
Organizers: R. McGeer, S. Malik

Session 20
Design for Testability

Chair: Yervant Zorian
Organizers: S. Kundu, J. Rajski

Session 27
Tutorial: How to Write AWK and Perl Scripts to Enable Your EDA Tools to Work Together

Session 28
Functional Verification Techniques

Session 29
Power Estimation

Session 30
Optimization of Sequential Circuits

Session 31
Topics in Physical Design

Chair: Lou Scheffer
Organizers: A. B. Kahng, A. Domic

Session 32
Consumer Product Design

Chair: Takayasu Sakurai
Organizers: T. Sakurai, S. Trimberger

Session 33
Tutorial: Issues and Answers in CAD Tool Interoperability

Chair: Mike Murray
Organizer: M. Murray
Presenters: *Mike Murray, Uwe B. Meding, Bill Berg, Yatin Trivedi, Bill McCaffrey, Ted Vucurevich* 509

Session 34
Hardware-Software Co-Design

Chair: Frank Vahid
Organizers: R. Gupta, L. Lavagno

Session 35
Timing and Power

Chair: Andrew T. Yang
Organizers: J. White, A. T. Yang

Session 44
Power Estimation and Retiming

Chair: Bill Lin
Organizers: F. Somenzi, B. Lin

Session 45
Technology Dependent Performance Driven Synthesis

Chair: Gabriele Saucier
Co-Chair: Hamid Savoj
Organizers: M. Pedram, G. Saucier

Session 46
Layout Analysis and Optimization

Chair: Alan Cave
Organizers: Y.-L. Lin, A. Domic

Physical Design CAD in Deep Sub-micron Era

Takashi Mitsuhashi, Takahiro Aoki, Masami Murakata,
and
Kenji Yoshida
Semiconductor DA & Test Engineering Center, TOSHIBA Corporation
580-1, Horikawa-cho, Saiwai-ku, Kawasaki 210, Japan

Abstract

In this paper, we will investigate the impacts of miniaturization of device dimensions that causes a paradigm shift in LSI design methodology. Major design issues in deep sub-micron LSIs, namely, wire delay, circuit complexity and power consumption will be discussed based on scaling theory. To resolve these issues, a concept called Layout Driven Synthesis and Optimization is introduced. Based on this concept, EDA programs including circuit optimizer, clock tree synthesis, technology mappers and so on, have been developed. Timing optimization and power minimization methods using the concept will be discussed in detail. Evaluation results obtained by proposed approach show superior performance and dramatic reduction of design period, and indicate validity of layout driven synthesis and optimization concept.

1 Introduction

As described by Dennard[1, 2], by adoption of ideal scaling with scaling parameter κ, geometrical dimensions and all voltages are reduced by $1/\kappa$. Impurity density of substrate is increased by κ. As a result, device density increases by κ^2 and power-dissipation density remains constant, and gate delay is reduced by $1/\kappa$. Following this scaling scenario, we can improve gate delay, increase device density, keep power-dissipation constant forever. This was a golden prophecy that promised the prosperity of today's LSI industries.

The reality is different from what is expected by the theory. Increasing power-dissipation is one of the most serious problem. One of the reason why we cannot achieve the constant power-dissipation density is that we cannot reduce the supply voltages. If the supply voltage remains constants, the power-dissipation increases by κ^3. The other reasons for increasing power-dissipation are increasing circuit complexity and operating frequency.

Wire delay is another serious problem. By ideal scaling, gate delay decreases by $1/\kappa$, but in reality, wire delay dose not decrease in such manner. One reason is that, because of the 3-D effect, the wire capacitance does not decrease as expected. The other reason is that, different from local interconnection, wire delay for global interconnection increases with increasing chip size. In conventional EDA paradigm, it was assumed that gate delay is dominant and wire delay can be negligible in higher level of design stages. As collapse of this paradigm, a serious "the chicken or the egg" type problem occurs. We need accurate wire delay for precise logic synthesis and even for high level synthesis. However, we cannot get accurate wire delay without layout that require netlist generated by logic synthesis, as input.

This problem was recognized in early '90s in logic optimization field, and several research efforts have been devoted. K. J. Singh et al.[3] and Yoshikawa et al.[4] proposed delay reduction methods that utilized the mapped netlist and improve it by local remapping, but layout is not considered explicitly. Pedram and Bhat [5, 6] proposed a technology mapping method that consider the wire delay and congestion by simulated placement of the circuit. Their approach is a pioneering work, but correlation between real placement and simulated one is more serious than expected. Lin et. al [7] proposed a gate sizing method that minimizes area under constraints of timing and delay, but actual placement is not considered. Kannan[8], Aoki et al.[9], and Sato et al.[10] proposed a logic optimization method based on layout. They claim 10 to 30% delay reduction. In other fields of EDA, this new paradigm becomes clear. Examples will be clock tree synthesis, RTL floorplanning, and so on. The important thing is that, for design optimization , we cannot do any thing without information based on layout. In this paper, post-layout logic optimization for timing

0-8186-7573-X/96 $5.00 © 1996 IEEE

and low power will be described, as an example.

2 Scaling Theory and Design Issues

Performance of Synchronous circuits, that is a major methodology in VLSI design, is determined by path delay between storage elements and skew in the clock distribution system. Maximum operating frequency of the circuit is determined by the worst path delay, not by average path delay. Timing optimization problem is formulated as to minimize the maximum path delay in the LSI. On the other hand, power dissipation is determined by sum of power dissipation of each gate in the LSI. Analysis of the interconnection length distribution in a logic chip[2] shows that interconnections consist of two groups, one is local interconnect and the other is global interconnect. These two groups show different statistical behavior with the scaling of dimensions.

Local interconnection: Applying ideal scaling by factor $\kappa(\kappa > 1)$to wire in LSI chip, wire width, thickness of metal, insulator thickness and even wire length become $1/\kappa$. As a result, resistance and capacitance of wire become κ and $1/\kappa$, respectively. This means RC delay cannot be improved by ideal scaling. To reduce the delay other scaling methods are proposed[2]. One example is to reduce wire width, metal thickness, and insulator thickness by $1/\sqrt{\kappa}$. By this scaling method, we can achieve RC delay scaling by $1/\kappa$.

Global interconnection: Different from local interconnection, wire length increases with $\sqrt{\kappa_A}$, where κ_A is the scaling factor for chip area. Scaling factor for resistance and capacitance are $\kappa^2\sqrt{\kappa_A}$, $\sqrt{\kappa_A}$, respectively. RC delay is described by $\kappa^2\kappa_A$. This means a drastic increase of global interconnection delay with scaling. Although several efforts to reduce RC delay of global interconnection are reported, the delay still increase as shown in Figure 1. Management of global interconnection is one of the most important design issue in deep sub-micron technology.

Power dissipation is another design problem in deep submicron VLSIs. Kuroda, et al[11]. reported a statistical data that power dissipations of MPUs and DSPs presented at ISSCC increase 4 times every 3 years.

The power dissipation of CMOS LSI is dominated by charge and discharge of load capacitance, although there is short circuit current component. Macroscopic power dissipation of a whole CMOS LSI chip is described by,

$$P_C = N \cdot \bar{C} \cdot f \cdot V_{DD}^2, \qquad (1)$$

where, N is the number of gates in the LSI, f is operating frequency, V_{DD}^2 is power supply voltage. $\bar{C}$ is average effective load capacitance, and described by,

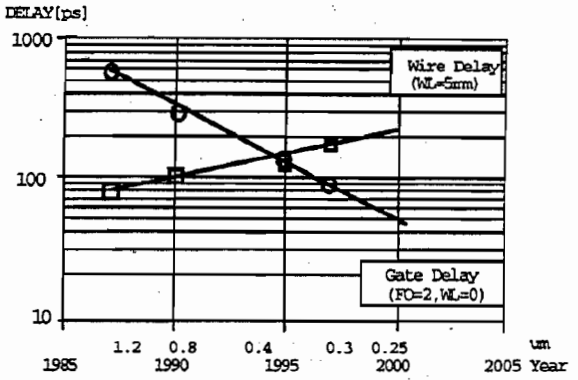


Figure 1: Wire Delay and Gate Delay

$$\bar{C} = \frac{1}{N}\sum_{i=1}^{N} C_L^i \cdot p_i, \qquad (2)$$

where C_L^i is load capacitance of gate i, p_i is switching probability of gate i in a clock cycle. Assuming changing rate per year of clock frequency α_f, scaling factor α_κ, power supply voltage $\alpha_{\kappa V}$, chip area κ_A, and power dissipation of current average LSIs P_0, power dissipation of average LSI after n year is described by,

$$P(n) = (\alpha_{\kappa_A}\alpha_\kappa\alpha_f\alpha_{\kappa V}^2)^n P_0 \qquad (3)$$

This equation(3) describes the trend of power dissipation, if trend parameters are given.

In this section, two critical issues in deep submicron LSI design have been discussed. One is wire delay, the other is power dissipation. In the following section, solutions based on layout driven synthesis and optimization will be described.

3 Layout Driven Optimization

In this section, a placement based net optimizer (PNO)[9, 12] is presented as an example of Layout Driven Optimization and Synthesis concept. PNO is a general purpose netlist optimizer based on layout information, that perform gate sizing, buffer insertion(Figure 2) and fanout decomposition(Figure 3). The program is applicable for timing optimization and minimization of power dissipation. The feature of the program is, that not only wire length for delay estimation but wire congestion and possibility of cell insertion on the target chip are considered.

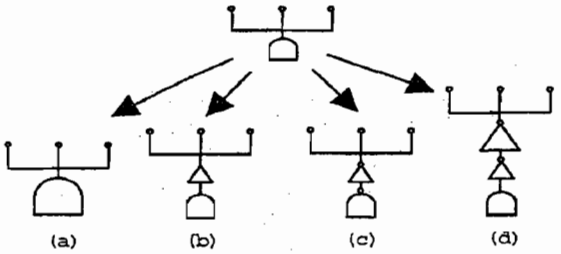


Figure 2: Repower:Gate Sizing and Buffer Insertion

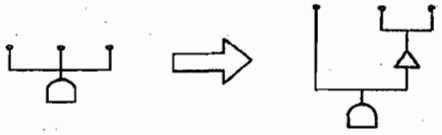


Figure 3: Fanout Decomposition and Buffer Insertion

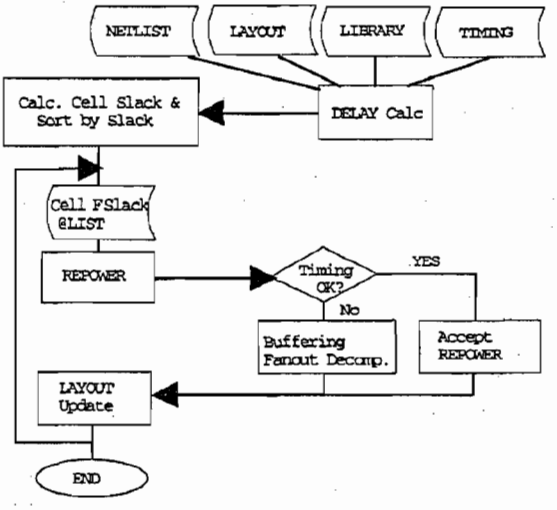


Figure 4: PNO: Circuit Optimization after Layout

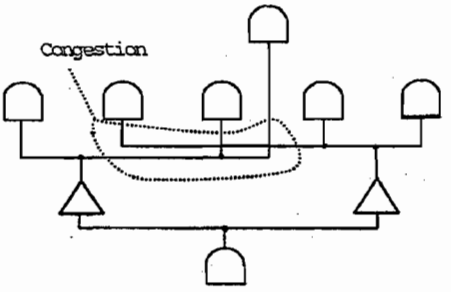


Figure 5: Wire Congestion by Fanout Decomposition

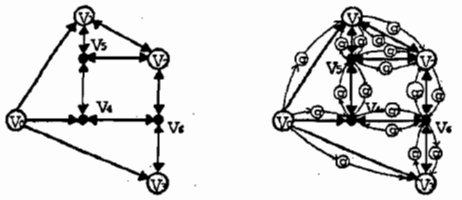
3.1 Timing Optimization

Placement based net optimizer(PNO) is an effective logic optimization program based on accurate delay value extracted from layout. PNO searches for a feasible solution, or a solution that satisfies timing constraints, started from an arbitrary initial solution. Figure 4 depicts the system configuration of PNO. Delay related to each cell is calculated using input data, namely Netlist, Layout, Library, and Timing Spec. Slack of each cell is calculated, and cells are sorted by its slack value. For each cell that does not satisfy the timing constraints, following operation is performed: (1) reduce delay by gate sizing and buffer insertion that is described in Figure 2, if timing spec. is satisfied go to next cell. (2) if timing is not satisfied, try fanout decomposition and buffer insertion as depicted in Figure 3. New cells that are generated by buffer insertion and gate sizing are placed on the original layout given as input. In this layout updating process, impact on given layout is minimized to avoid bad influence on the chip performance.

Problems we must take care in fanout decomposition and buffer insertion, are as follows: (1) Some kinds of fanout decomposition increase wire congestion as shown in Figure 5. Decomposition program is required not to increase wire congestion. (2) Optimal placement of the inserted buffer is not guaranteed, because of less considerration of layout status of the chip. (3) Accurate delay estimation based on wiring resistance and capacitance is essential.

To cope with above mentioned problems, a network model that represent cell positon and wiring route as exact as possible is adopted. The network model consists of a set of vertices representing cells and Steiner points, and a set of edges representing possible wire routes. Steiner point is a vertex on a graph, that represents a possible branching point without cells. The network model is embedded in a plane that represents chip area. Each cell and Steiner point is mapped to a position where those geometrical objects are placed. Edges are also embedded in a plane where possible routes are available. Possible buffer insertion points where vacancy is available are also added to the model as edge vertex sets.

Figure 6 is an example of network model. In this graph, a set of vertices $V_C = \{V_0, V_1, V_2, V_3\}$ represents cells, $V_S = \{V_4, V_5, V_6\}$ represents Steiner points, and $V_B = \{g, g_1, g_2\}$ represents possible buffer insertion point, respectively. The edge set means possible wiring route. Direction is given to each edge, edges connected to signal source are directed from signal source to sink. Other edges are bi-directional.



(a) : Embedded Route Graph (b) : Fanout Graph

Figure 6: Network Model Generation

The network model is generated by two steps. In first step, route graph $G_R(V_R, E_R)$ is generated, where cell vertex set V_C represents cells given as the problem. XY-coordinate of $v_i \in V_C$ is the position of $Cell_i$. Steiner points V_S and edge is generated as follows: (1) Let signal source vertex be origin vertex. (2) Find the nearest vertex $v_i \in V_C$ in each quadrant from the origin vertex. (3) Generate edges that connect the origin and the nearest vertex in each quadrant. (4) Let the nearest vertex in each quadrant be next origin vertices. (5) Repeate (2) - (4), until all vertices are connected. (6) Make a bounding box for each generated edge. Newly generated vertices of the bounding box are treated as Steiner points and added to V_S. The new edges are added to E_R.

In the second step, fanout graph $G_F(V_F, E_F)$ is generated. Fanout graph is a model used for determining oiptimal tree structure and buffer insertion points. The procedure check each edge and corresponding region on a chip. If there is vacant space in the placement near the region, the program generate a buffer vertex and edges parallel to the edge under checking.

Delay calculation for this problem is interesting, the delay value is structure dependent and strongly influenced by buffer insertion. This nature of the delay calculation, we adopted a dynamic delay calculation method, that calculate the delay $d(v_0, v_t)$ from root v_0 to vertex v_t based on the value $d(v_0, v_{t-1})$,

$$d_a(v_0, v_t) = d_a(v_0, v_{t-1}) + \beta R(v_{t-1}, v_t) C(v_{t-1}, v_t) + \alpha R(v_{t-1}, v_t) C_i(v_t), \quad (4)$$

$$d_a(;v_t) = \alpha R(v_0) C_l(v_0), \quad (5)$$

where α and β are constants, and determined based on delay type to be calculated. $C_i(v_t)$ is load capacitance connected to terminal v_t, $C(v_{t-1}, v_t)$ and $R(v_{t-1}, v_t)$ are capacitance and resistance value for wire segment between v_{t-1} and v_t. These values can be calculated by estimated wire route or Manhattan wire length. $R(v_i)$ is the on-resistance at vertex v_i, $C_l(v_i)$ is total capacitance from vertex v_i to leaves.

If new vertex v_{t+1} is added to the tree at v_t, the delay from v_0 to v_{t+1} is represented as,

$$d_a(v_0, v_{t+1}) = d_a(v_0, v_t) + e(v_t, v_{t+1}), \quad (6)$$

where,

$$e(v_t, v_{t+1}) = C_\delta(\beta R(v_{t-1}, v_t) + \alpha R_m(v_t)) + \beta R(v_t, v_{t+1}) C(v_t, v_{t+1}) + \alpha R(v_t, v_{t+1}) C_i(v_{t+1}),$$
$$R_m(v_t) = R(v_0) + ... + R(v_{t-1}) + R(v_t),$$
$$C_\delta = C(v_t, v_{t+1}) + C_i(v_1).$$

$e(v_t, v_{t+1})$ can be interpreted as the increase of delay by adding a new vertex v_{t+1} to the tree. When buffer is inserted, a special treatment is required for delay calculation. If there is a buffer in a path, the path delay is simple sum of delay from signal source to buffer input and buffer to net pin.

A^*-search algorithm[14] is used to find a optimal buffer insertion points and generate optimal fanout decomposition. The algorithm is efficient for finding optimal path between two vertices s and v. Usually, following objective function is used for A^*-search,

$$\hat{f}(v) = \hat{a}(v) + \hat{e}(v),$$

where $\hat{a}(v)$ gives estimated cost for optimal path between start vertex s and v, $\hat{e}(v)$ gives estimated cost for optimal path between v and terget vertex t. Compared this equation with equation(6), $d_a(v_0, v_t)$ corresponds $\hat{a}(v)$ and $e(v_t, v_{t+1})$ corresponds $\hat{e}(v)$.

Fanout decomposition and buffer insertion problem is a tree generation problem with timing constraints. The problem is stated as follows: Given a set of timing constraints from signal source to pins, find a optimal tree structure and buffer insertion points. The search is performed as follows: Let N_t be a set of subtree vertex, that is already determined and H be a heap that keeps current vertices for expanding the tree. Procedure is as follows: (1) Get vertex $u \in H$. (2) Evaluate cost for vertex v which is adjacent to u. (3) If v has not trace-back points, return v to H with evaluated value. (4) If v has trace-back points, and new cost is better, replace by new trace-back points and cost. (5) Repeat until the target vertex v^* is obtained from H.

3.2 Power minimization

Placement based net optimizer(PNO) is applicable to minimization of power dissipation. Convensionally power minimization procedure is considered as follows: (1) Find a circuit that satisfy the every timing constraints, (2) minimize the power dissipation under

Table 1: Comparison of two low power design method

	Without L.P.	Conven.	Proposed
$I_{DD}(mA)$	94.2	92.5	22.0
$Delay(nS)$	15.0	15.0	15.0

constraints of timing. We have proposed a new and effective procedure[12] for power minimization, that is described as follows: (1) change every cell to the smallest size without considering timing constraints, (2)then improve critical path by PNO. Comparison of two methods shows that the new procedure yields good results.

Major power dissipation of LSI consists of charge and discharge current of wiring capacitance I_w, cell input pin capacitance I_{pin}, parastic capacitance in cell I_{cell}, and short circuit current I_s. The method describrd here can reduce I_{pin}, I_{cell}, and I_s. I_{pin} and I_{cell} can be reduced by using smaller dimension cells. Short circuit current I_s depends on slope of input waveform and load capacitance of the gate. Excess I_s can be privented by setting limitation on load capacitance.

Above mentioned two methods are compared by bench mark test using an LSI circuit that consists of 32kcell and 34knet. The device technology is $0.5\mu m$. Results shown in table 1 indicates that proposed method gives good results and preferable. Without Layout Driven Optimization and Synthesis concept, It is difficult for us to perform such a drastic power dissipation by EDA programs.

4 Experimental Results

In this section, we will present some experimental results that validate the effectiveness of layout driven synthesis and optimization approach. A placement based net optimizer(PNO), that is an engine for timing optimization and power minimization, is implemented on SUN/Sparc 10. Bench mark results of both timing optimization and power minimization will be described, respectively.

Timing optimization - procedure for delay reduction is as follows: (1) placement by timing driven layout[13], (2) timing optimization by PNO. In step (2), critical paths are recognized and improved by PNO. As shown in Table 2[9], 8 circuits were selected for evaluation. Path delay before timing optimization

Table 2: Timing Improvement by PNO

CKT	Tech.	#	#	Delay(nSec)		Imprv.
	μm	Net	Cell	Before	After	(%)
chip-1	0.5	60k	50k	23.00	18.98	17.5
				17.49	11.14	36.3
				15.74	11.14	29.1
				7.95	7.39	7.7
chip-2	0.8	23k	20k	50.09	41.99	16.2
				47.28	39.60	16.2
				44.61	35.75	19.9
				43.72	36.29	17.0
chip-3	0.8	54k	43k	52.44	44.77	14.8
				27.78	24.33	12.4
chip-4	0.8	23k	18k	42.87	39.88	7.0
				32.13	18.44	42.6
				22.53	16.52	26.7
				18.14	14.72	18.7
				17.81	12.99	27.1
				5.26	5.14	2.3
chip-5	0.8	14k	11k	10.88	10.81	0.6
				41.88	39.00	6.9
chip-6	0.5	12k	12k	14.87	13.68	8.1
chip-7	0.5	19k	17k	15.65	12.44	20.5
chip-8	0.5	30k	26k	13.73	11.54	15.9

is indicated in column "Before", delay after optimization is displayed under "After". 17% delay reduction in average is observed. CPU time required for PNO is evaluated for chip-6, -7, and -8, and reported 15.3min., 28.5min., and 17.5min., respectively.

Power minimization - procedure for power minimization is: (1) placement by timing driven layout, (2) change all cells into minimal dimension, and (3) improve path delays that violate timing constraints by PNO. As shown in Figure 7, two classes of sizing have been adopted. One is usual gate sizing, the other is F/F sizing. In usual gate sizing, flip-flops are not changed by optimizer. In F/F sizing, flip-flops with smaller size transistors associated with output buffer is replaced as a storage element. The size of output buffers are optimized by PNO. By virtue of small F/F, power dissipation by clock is reduced significantly. Flexible structure allows optimal selection of buffer size depending on the delay.

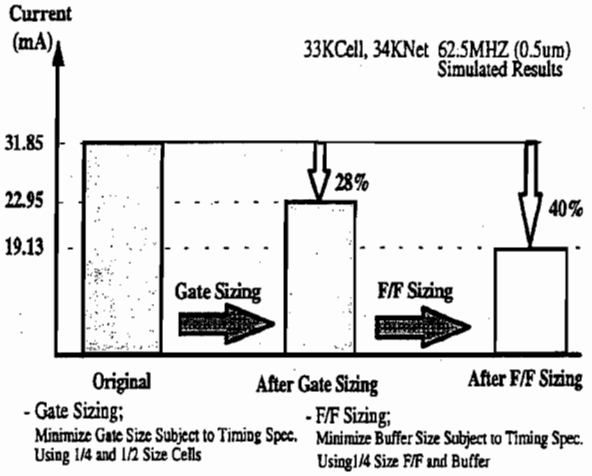


Figure 7: Power Reduction by Gate Sizing and F/F Sizing

5 Conclusions

A concept called layout driven synthesis and optimization that is indispensable for deep sub-micron LSI design is proposed. Power dissipation and wiring delay issues are discussed. These are predicted to be serious problems, if device feature size of LSI is reduced along with ideal scaling theory. We also pointed out that, if the wire delays of global interconnections becomes dominant, EDA paradigm will be required to change. High level or logic level designs based on conventional estimated delay becomes meaningless, and layout driven synthesis and optimization concept will be required. To validate this idea, we have made two experiments; timing optimization, and minimization of power dissipation. Placement based net optimizer was used as optimization engine in both cases. Results obtained by proposed approach show superior performance, and indicate validity of the layout driven synthesis and optimization concept.

References

[1] R.H. Dennard, F.H. Gaensslen, H.N. Yu, V.L. Rideout, E. Bassous, and A.R. LeBlanc, "Design of ion implanted MOSFET's with very small physical dimensions," IEEE J. of Solid-State Circuits, Vol.SC-9,pp.256-268, Oct.1974.

[2] H.B.Bakoglu, "Circuits, Interconnections, and Packaging for VLSI", Addison-Wesley,1990.

[3] K.J.Singh and A. Sangiovanni-Vincentelli, "A Heuristic Algorithm for the Fanout Problem", 27th DAC Proc., pp.357-360, 1990.

[4] K.Yoshikawa, H.Ichiryu, H.Tanishita, S.Suzuki, N.Nomizu, and A.Kondoh, "Timing Optimization on Mapped Circuits", 28th DAC Proc., pp.112-117, 1991.

[5] M.Pedram and N.Bhat, "Layout Driven Technology Mapping", 28th DAC Proc., pp.99-105, 1991.

[6] M.Pedram and N.Bhat, "Layout Driven Logic Restructuring/Decomposition", 1991 ICCAD Proc., pp.134-137, 1991.

[7] S.Lin, M.Marek-Sadoeska, E.S.Kuh, "Dealy and Area Optimization in Standard-Cell Design", 27th DAC Proc., pp.349-352, 1990.

[8] L.N.Kannan, P.R.Suaris, and H-G. Fang, "A Methodology and Algorithms for Post-Placement Delay Optimization", 31st DAC Proc., pp.327-332, 1994.

[9] T.Aoki, M.Murakata, T.Mitsuhashi and N.Goto, "Fanout-tree Restructuring Algorithm for Post-placement Timing Optimization", Proc. of the ASP-DAC '95, pp.417-422, 1995.

[10] K.Sato, M. Kawarabayashi, H. Emura, and N.Maeda, "Post-layout Optimization for Deep Submicron Design", Proc. of the 33rd DAC, pp.740-745, 1996.

[11] T.Kuroda and T.Sakurai, "A new guideline for low-power circuit design", A Technical White Paper on Low Power LSIs(in Japanese), Nikkei Business Publications, Inc.,Tokyo, 1994.

[12] T.Aoki, F.Minami, and M.Murakata, "A gate sizing and buffer insertion for power reduction", (in Japanese), Proc. of IEICE General Conference '96, SA-2, Sept., 1996.

[13] M.Murakata, M.Murofushi, M.Igarashi, T.Aoki, T.Ishioka, T.Mitsuhashi, and N.Goto, "Concurrent Logic and Layout Design System for High Performance LSIs", Proc. of 1995 CICC, pp.465-468, 1995.

[14] N.J.Nilsson, "Problem-Solving Methods in Artificial Intelligence", New York, McGraw-Hill, 1971.

TAB 35

Proceedings of the

ASP-DAC'95/CHDL'95/VLSI'95

Asia and South Pacific Design Automation Conference

IFIP International Conference on Computer Hardware Description Languages and their Applications

IFIP International Conference on Very Large Scale Integration

August 29–September 1, 1995

Makuhari Messe, Chiba, Japan

Sponsored by

IFIP WG10.5 (Former 10.2 and 10.5)
IEICE (Institute of Electronics, Information and Communication Engineers)
IPSJ (Information Processing Society of Japan)
ACM SIGDA
IEEE Circuits and Systems Society
IEEE Computer Society

ASP-DAC'95/CHDL'95/VLSI'95 Proceedings

IEEE Catalog Number: 95TH8102
ISBN 4-930813-67-0 Softbound Edition
0-7803-2763-2 Microfiche Edition
Library of Congress N/A

Supported in part by the Ministry of Education, Science and Culture under the Grant-in-Aid for Publication of Scientific Research Result.

Table of Contents

Fanout-tree Restructuring Algorithm for Post-placement Timing Optimization

T.Aoki, M.Murakata, T.Mitsuhashi and N.Goto†
Semiconductor DA & Test Engineering Center,
†Research and Development Center,
TOSHIBA
Kawasaki, JAPAN

Abstract

This paper proposes a *fanout-tree restructuring* algorithm for post-placement timing optimization to meet timing constraints. The proposed algorithm restructures a fanout-tree by finding a tree in a graph which represents a multi-terminal net, and inserts buffer cells and resizes cells based on an accurate interconnection RC delay without degrading routability. The algorithm has been implemented and applied to a number of layout data generated by timing driven placement. Application results show a 17% reduction in circuit delay on the average.

1 Introduction

With today's sub-micron process, the interconnection delay becomes a dominant factor in total circuit delay. Therefore, timing consideration in logic and layout design stages is indispensable for designing high performance LSIs.

A number of works on repower of cell size and buffer insertion have been proposed [5, 6, 7, 8, 9]. However, some traditional methods may arise timing violations after layout because the methods use inaccurate delay based on the number of fanouts.

To solve this problem, several synthesis/resynthesis methods that consider layout information in the synthesis process have been presented. Pedram et al. [1, 2] describe a logic synthesis method in which rough placement is done during logic synthesis process, and use placement information to estimate delay and routability. Ginetti et al. [3] and Kannan et al. [4] describe a gate sizing and fanout-buffering method, which uses a placement information to calculate interconnection delay. However, timing measures used in these methods are based on the estimated wire load capacitance. So, the estimated interconnection delays during the synthesis process are very different from the actual interconnection delays after placement. For today's sub-micron process, RC delay can amount to as much as 15% of the total interconnection delay. To obtain good results, or to obtain timing error free design, it is important to optimize circuits on interconnection RC delay based on the placement.

In this paper, we present a fanout-tree restructuring algorithm for post-placement timing optimization. The proposed algorithm restructures a fanout-tree by finding a tree in a graph that represents a multi-terminal net. Furthermore, the method inserts buffer cells and resize cells based on an accurate interconnection RC delay without degrading routability. This method is incorporated into a concurrent logic and layout design system [10] to reduce circuit delay on the layout stage. The algorithm has been implemented and applied to a number of layout data generated by timing driven placement. Application results show a 17% reduction in circuit delay on the average.

The paper is organized as follows: Section 2 introduces a basic idea for our approach. Some definitions and problem formulation are shown in Section 3. The detail of the fanout-tree_restructuring algorithm is described in Section 4. The outline of the post-placement timing optimization is described in Section 5. Experimental results are shown in Section 6 and concluding remarks are given in Section 7.

2 Basic Idea

In this section, we describe a basic idea of the proposed *fanout-tree restructuring* algorithm. Fig. 1 is an example of tree structure of a multi-terminal net, v_0 is a signal source, $v_1, \ldots, v_3$ are signal sink (v_1 isn't connected with the tree yet) and v_6 is a Steiner point. In this example, an interconnection delay from v_0 to v_2 can be expressed as follows [11]:

$$\begin{aligned} d_a(v_0, v_2) &= d_a(v_0, v_6) + \beta R(v_6, v_2) C(v_6, v_2) \\ &\quad + \alpha R(v_6, v_2) C_i(v_2), \\ d_a(\cdot, v_0) &= \alpha R(v_0) C_l(v_0), \end{aligned} \tag{1}$$

where $\alpha = 1.1$, $\beta = 0.7$, $C_i(v_i)$ is the input capacitance of terminal v_i, $C(v_i, v_j)$ is the wire segment capacitance between terminal v_i and v_j, and $R(v_i, v_j)$ is a wire segment resistance, $R(v_i)$ is a on-resistance of source terminal v_i and $C_l(v_i)$ is the load capacitance from vertex v_i toward leaf of a tree. $C(v_i, v_j)$ and $R(v_i, v_j)$ are expressed as follows:

$$C(v_i, v_j) = cL(v_i, v_j), \quad (2)$$
$$R(v_i, v_j) = rL(v_i, v_j), \quad (3)$$
$$L(v_i, v_j) = |x(v_i) - x(v_j)| + |y(v_i) - y(v_j)|,$$

where c is the wire capacitance per unit length, r is the wire resistance, $L(v_i, v_j)$ is a rectilinear distance between v_i and v_j, and $x(v_i)$ and $y(v_i)$ are x coordinate and y coordinate, respectively.

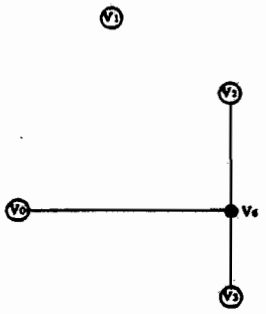


Figure 1: Example of routing tree. v_1 isn't connected with this tree yet.

Let's consider the case that the new signal sink terminal v_1 is connected to terminal v_2. The new delay from v_0 to v_1 can be calculated as follows:

$$d_a(v_0, v_1) = d_a(v_0, v_2) + e(v_2, v_1), \quad (4)$$
$$e(v_2, v_1) = C_\delta(\beta R(v_6, v_2) + \alpha R_m(v_2)) + \beta R(v_2, v_1) C(v_2, v_1) + \alpha R(v_2, v_1) C_i(v_1),$$
$$R_m(v_2) = R(v_0) + R(v_0, v_6) + R(v_6, v_2),$$
$$C_\delta = C(v_2, v_1) + C_i(v_1),$$

where $d_a(v_0, v_2)$ is given by (1), $R_m(v_2)$ is a sum of wire segment resistance from terminal v_0 to v_2 and C_δ is the increase of load capacitance. In equation (4), $d_a(v_0, v_2)$ is the actual delay from v_0 to v_2 and $e(v_2, v_1)$ is a estimation delay from v_2 to v_1.

We use A*-Search algorithm to find buffer insertion points. The algorithm is known as the efficient algorithm to find the minimum/maximum cost paths from a given vertex s to other vertex t in a graph by the following estimation cost $\hat{f}(v)$.

$$\hat{f}(v) = \hat{a}(v) + \hat{e}(v), \quad (5)$$

where $\hat{a}(v)$ is a estimation cost of the optimal path from s to v. Also, $\hat{e}(v)$ is a estimation cost from v to t. In equation (4), the term $d_a(v_0, v_2)$ corresponds to $\hat{a}(v)$ and $e(v_2, v_1)$ corresponds to $\hat{e}(v)$. Then, the delay minimization problem can be solved by the A*-Search.

The *fanout-tree restructuring* algorithm is based on the above idea. It composed of the following basic two steps. In the first step, a *fanout graph* is generated from a multi-terminal net (as shown in Fig. 6). The detail of the *fanout graph* is described in Section 4.1. The *fanout graph* includes terminals, Steiner points of routing tree and buffer cells that will be inserted by *fanout-tree restructuring* process. In the second step, fanout-tree is generated by finding a path based on A*-Search. The generated fanout-tree suggests locations for the point of newly inserted buffer cells and its size. The detail of new estimation cost for A*-Search and path finding algorithm is described in Section 4.2.

3 Definitions

In this section, we introduce some definitions that are used for describing the algorithm.

Let c be a cell in a circuit C. Each cell has a set of signal sink terminals, or $\mathcal{I}(c)$, and a source terminal, or $v_0(c)$. The source terminal $v_0(c)$ has a connection to a set of sink terminals of other cells that is called *fanout set* of $v_0(c)$, or $\mathcal{F}(v_0(c))$. A terminal which drives $t \in \mathcal{I}(c)$ is denoted by $\mathcal{D}(t)$. Fig. 2 shows an example, $\mathcal{F}(v_0) = \{v_1, v_2, v_3\}$, $\mathcal{I}(c) = \{t_1^I, t_2^I\}$, $\mathcal{D}(t_1) = t_4^I$ and $\mathcal{D}(t_2) = t_5^I$.

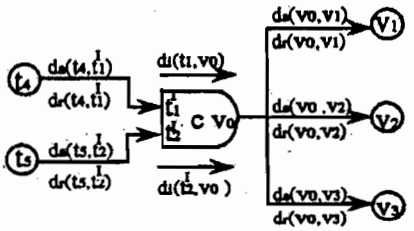


Figure 2: Calculating the actual and required delay.

A timing constraint $d_r(v_i)$ and an actual delay $d_a(v_i)$ for each terminal v_i belong to *fanout set* can be expressed as follows:

$$d_r(v_i) = d_r(v_0(c), v_i) + MIN[d_i(t_j^I, v_0(c)) + d_r(\mathcal{D}(t_j^I), t_j^I) | t_j^I \in \mathcal{I}(c), v_i \in \mathcal{F}(v_0(c))], \quad (6)$$
$$d_a(v_i) = d_a(v_0(c), v_i) + MAX[d_i(t_j^I, v_0(c)) + d_a(\mathcal{D}(t_j^I), t_j^I) | t_j^I \in \mathcal{I}(c), v_i \in \mathcal{F}(v_0(c))], \quad (7)$$

where $d_i(t_j^I, v_0(c))$ is an intrinsic delay from t_j^I to $v_0(c)$

$d_r(v_0(c), v_i)$ and $d_a(v_0(c), v_i)$ are timing requirement and actual interconnection delay from $v_0(c)$ to v_i, respectively. As equation (6) and (7), a slack for cell c, or $z(c)$ can be expressed as follows:

$$\begin{aligned} z(c) &= z_r(c) - z_a(c), \qquad (8) \\ z_r(c) &= MIN[d_r(v_i)|v_i \in \mathcal{F}(v_0(c))], \\ z_a(c) &= MAX[d_a(v_i)|v_i \in \mathcal{F}(v_0(c))]. \end{aligned}$$

The problem of fanout-tree restructuring is formulated as follows:

minimize $|z(c)|, c \in C$
subject to $d_r(\mathcal{D}(t_i^I), t_i^I) \geq d_a(\mathcal{D}(t_i^I), t_i^I), t_i^I \in \mathcal{I}(()c)$.

4 Fanout-Tree Restructuring Algorithm

In this section, we introduce the *fanout-tree restructuring* algorithm. This algorithm consists of two basic transformations, *repower* and *buffering*. The transformations are illustrated in Fig. 3 and Fig. 4. The *repower* chooses the best size of cell c and/or inserts buffer b (Fig. 3). The *buffering* inserts non-inverting buffers ($b1$) and remakes a topology of the multi-terminal net (Fig. 4).

The following pseudo-code is the *fanout-tree restructuring* algorithm. In the first step, the algorithm evaluates the slack $z(c)$ for the result of *repower*. If *repower* does not meet timing constraint ($z(c) < 0$), the algorithm applies *buffering*. The *buffering* process makes a *fanout graph*, or $G'(V', E')$ and finds the tree N_t in the G' by repeating a path search. The found tree expresses how to remake a topology of the multi-terminal net.

```
Fanout-Tree-Restructuring(c,ξ,τ,λ)
{
  z(c) = Repower(c,ξ,τ,λ);
  if(z(c) ≤ 0) return;
  G'(V', E') = Make-Fanout-Graph(c,ξ,λ);
  N_t = source vertex;
  Let SINK be a set of all the sink vertices of V'
      which is sorted by the slack.
  for each v* ∈ SINK {
      if(v* is included N_t) continue;
      PATH = A*-Search(N_t, v*, τ);
      N_t = N_t ∪ PATH;
      Update-Tree-Spec(N_t); }
  Insert-Buffer(N_t, ξ) ;
}
```

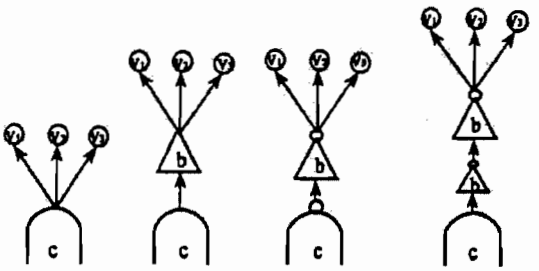


Figure 3: Repower transformations

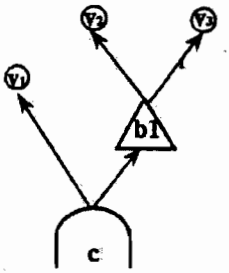


Figure 4: Buffering transformations

4.1 Fanout Graph

In this subsection, we define the *fanout graph* which is used in *buffering* to generate a fanout-tree.

A multi-terminal net can be represented as a directed planar graph (DPG), or $G(V, E)$. Where V is a set of vertices which correspond to a signal source terminal v_0 (*source*), sink terminals $\mathcal{F}(v_0)$ (*sink*) and Steiner points W (*spoint*), that is $V = \{v_0\} \cup \mathcal{F}(v_0) \cup W$. Also, E is a set of directed edge. If v_i and v_j can be connected by straight line and v_j isn't *source*, v_i has a outgoing edge from v_i to v_j, or $(v_i, v_j) \in E$. In DPG, there is a *source* at most one and it has only outgoing edges. Fig. 5 shows an example of DPG, where v_0 is a *source*, $v_1, \ldots, v_3$ are *sink* and $v_4, \ldots, v_6$ are *spoint*. The symbol '$\rightarrow$' are uni-directed edges and '$\leftrightarrow$' are bi-directed edges.

Let $G'(V', E')$ be a *fanout graph* that is made by inserting new vertices and edges into a DPG $G(V, E)$, that is $V \subset V'$ and $E \subset E'$. If there is an edge $(v_i, v_j) \in E$, we insert a new vertex g which corresponds to a buffer cell and insert new two edges (v_i, g) and (g, v_j).

The following pseudo-code is the algorithm to make *fanout graph*.

```
Make-Fanout-Graph(c,ξ,λ)
{
  E' = ∅, V' = V;
  G(V, E) = Make-DPG(c,ξ,λ);
  for each (v_i, v_j) ∈ E {
      [x(g), y(g)] = SearchLoc(gξ);
```

```
        if(g can't place) continue;
        V' = V' ∪ {g}, E' = E' ∪ {(v_i, g), (g, v_j)};}
    return G'(V', E');
}
```

The *SearchLoc* step aims to search a rectangular area $([x(v_i), y(v_i)], [x(v_j), y(v_j)])$ for a placement location of g. Fig. 6 depicts the *fanout graph* G' that is re-made from DPG in Fig. 5, where g, g_1, g_2 are inserted *buffer* .

4.2 Path Search

In order to finds the path which meet the timing constraint, we use the A*-Search technique [13]. This approach is based on the performance driven global router described in [14]. In the first step of A*-Search, start vertices are pushed into a heap list H, that is maintained during path search. The A*-Search repeats the following process until the target vertex v^* is popped out from H.

1. Pop out a vertex u from H.
2. Calculate new cost for vertex v which adjacent to u.
3. If vertex v does not have trace-back-point, give cost and push v into H.
4. If v has trace-back-point, compare the new cost and old cost. If new cost are better than old cost, change new trace-back-point of v to u, give new cost to v and push v into H.

To find the path which meet the timing constraint, we define following cost function from equation (5).

$$
\begin{aligned}
f(v) &= d_r(v^*) - (d_a(v) + e(v)), \\
d_a(v) &= d_a(u) + \alpha R_m(u)(C(u,v) + C_i(v) \\
&\quad + \beta C(u,v)R(u,v) + \alpha R(u,v)C_i(v), \\
e(v) &= \alpha R_m(u) C_{buf} + \alpha R_{buf}[C(v,v^*) + C_i(v^*)] \\
&\quad + \beta C(v,v^*)C_i(v^*) + \alpha C(v,v^*)C_i(v^*), \\
R_m(v) &= \begin{cases} \textit{on-resistance} & \text{if } v \text{ is } \textit{buffer} \\ R_m(u) + R(u,v) & \text{otherwise} \end{cases} \\
C_m(v) &= MIN[C_m(u) - C(u,v) - C_i(v), \\
&\quad (d_r(v) - d_a(u,v))/R_m(u)], \\
(u,v) &\in E',
\end{aligned}
$$

where C_{buf} and R_{buf} are input capacitance and on-resistance of *buffer* , respectively. $C(u,v)$ and $R(u,v)$ are given by equation (2) and (3). $R_m(v)$ is a sum of interconnection resistance from *source* $v_0(c)$ to v. $d_r(v^*)$ is a required delay for a target vertex $v^* \in \mathcal{F}(v_0(c))$. $e(v)$ is a estimated delay from v to target v^*. $d_a(v)$ is an actual delay from *source* $v_0(c)$ to v when passing through u. $C_m(v)$ is an upper bound of add-able capacitance at v to meet timing constraints of already found paths.

In the step 4, if the value of $d_a(v)$ is bigger than required delay $d_r(v)$, the A*-Search does not push v into H because the condition $d_a(v) > d_r(v)$ means that the delay of this path does not meet the timing constraint for v. Also, the value of $C_m(v)$ is a negative, the A*-Search does not push v into Hbecause the negative value of $C_m(v)$ means that we can not add any capacitance to v.

In the stepl, the target vertex v^* is popped out from H, the A*-Search traces back from the target v^* to start vertices to fix the path.

Let us consider a case of path search with an example of Fig. 6, Fig. 7 and Fig. 8. We assume that Fig. 6 is a fanout graph and Fig. 7 is a tree that was found so far. In this case, the set of start vertex is $N_t = \{v_0, v_1, v_3, v_4, v_5, v_6, g_1\}$, the target vertex is $v^* = v_2$. The final step traces back the path as $(v_2 \rightarrow g_2 \rightarrow v_6)$. As a result, the path $PATH = \{v_2, g_2, v_6\}$ is found. In this example, the net topology is depicted in Fig. 9.

4.3 Repower

In this subsection, we describe the algorithm of *repower* . The *repower* has done following transformations:

- Substitute other cell that has the same boolean function.
- Substitute other cell that has a negative boolean function and insert inverting buffer.
- Insert a non-inverting buffer.
- Insert two inverting buffers.

These transformations are depicted in Fig. 3. The *repower* algorithm is simple. The first step of *repower* makes a candidate list of transformation. The transformation candidate for a cell c is a combination of logically equivalent/negative cells and inverting/non-inverting buffers. In the second step, *repower* has done the following processes for each candidate of transformation:

- Place the substituted cell and inserted buffers.
- Construct a *trunk Steiner tree* [12] for calculating objective function $z(c)$.
- Choose the best transformation that has the minimum $z(c)$.

5 Post-placement Net-list Optimization

In this section, we describe the inputs, outputs and an outline of the post-placement net-list optimization (*PNO*). The inputs are a net-list of a circuit C, after placement layout data ξ, timing constraints τ and library λ. The input library includes multiple transistor size cells which have the same boolean function and the timing information of cell consists of an intrinsic delay, input capacitance and on-resistance. The input timing constraint consists required time for each interconnection. The following pseudo-code is an outline of *PNO*.

```
postPlacementNetlistOptimizer(C,ξ,τ,λ)
{
  CELL_LIST = ∅;
  for echo c ∈ C {
      z(c) = Calculate-Slack(c, τ, ξ);
      CELL_LIST ← [c, z(c)]; }
  for each c ∈ CELL_LIST
      Fanout-Tree-Restructuring(c,ξ,τ,λ);
}
```

In the first step of the *PNO*, required and actual delay for each cell c is calculated. The *CELL_LIST* is sorted by slack $z(c)$, so that the first cell of the list has the smallest slack. In the second step, the *fanout-tree restructuring* , which resizes a cell and inserts buffers, is applied to each cell in *CELL_LIST* to meet timing constraints. The outputs are net-list and layout data. The output net-list is optimized by the *fanout-tree restructuring* . Also, the output layout data has new location of resized cells and inserted buffers.

6 Experimental Result

The above method has been implemented. The program runs on SUN/SparcStation-10 and applied to SOG/EA type LSIs as shown in Table 1. Experiments have been performed as the following procedure: (1) Timing driven placement with an initial synthesized net-list. (2) Net-list optimization by our method(*PNO*) for the timing driven placement result. Set the timing constraints to critical paths based on the precisely estimated interconnection delays for given cell placement results [10].

The experimental results are shown in Table 2. The column labeled *Delay* represents the critical path delays. Notice that *pre* is a before net-list optimization and *post* is an after optimization. On the average, delay was reduced by 17%.

Table 1: Chips characteristics

	CHIP1	CHIP2	CHIP3	CHIP4
Technology	0.5μm	0.8μm	0.8μm	0.8μm
#Nets/#Cells	60k/50k	23k/20k	54k/43k	23k/18k
	CHIP5	CHIP6	CHIP7	CHIP7
Technology	0.8μm	0.5μm	0.5μm	0.5μm
#Nets/#Cells	14k/11k	12k/12k	19k/17k	30k/26k

Table 2: Experimental results

	Delay(ns)		delay	cpu
	pre	post	reduction(%)	time(min:sec)
CHIP1	23.00	18.98	17.5	
	17.49	11.14	36.3	
	15.74	11.14	29.1	
	7.95	7.39	7.7	
CHIP2	50.09	41.99	16.2	
	47.28	39.60	16.2	
	44.61	35.75	19.9	
	43.72	36.29	17.0	
CHIP3	52.44	44.77	14.8	
	27.78	24.33	12.4	
CHIP4	42.87	39.88	7.0	
	32.13	18.44	42.6	
	22.53	16.52	26.7	
	18.14	14.72	18.7	
	17.81	12.99	27.1	
	5.26	5.14	2.3	
CHIP5	10.88	10.81	0.6	
	41.88	39.00	6.9	
CHIP6	14.87	13.68	8.1	15:19
CHIP7	15.65	12.44	20.5	28:31
CHIP7	13.73	11.54	15.9	17:29

7 Conclusions

We have presented a *fanout-tree restructuring* algorithm for post-placement timing optimization. The algorithm performs the restructuring to meet the timing constraints by buffer insertion and gate sizing. By the A*-Search on fanout graph, the optimal buffer locations are found without degrading routability that may occur by wire crossing caused by the restructuring. Experimental results show 17% reduction in circuit timing for circuits laid out by timing driven placement program.

References

[1] M.Pedram and N.Bhat, *28th DAC*, pp.99-105,1991.

[2] M.Pedram and N.Bhat, *ICCAD-91*, pp.134-137,1991.

[3] A.Ginetti and D.Brasen, *CICC-93*, pp.9.2.1-9.2.4,1993.

[4] L.N.Kannan,P.R.Suaris and H.Fang, *31st DAC*, pp.327-332,1994.

[5] M.A.Cirit, *24th DAC*, pp.121-123,1987.

[6] P.K.Chan. *27th DAC*, pp.353-356,1990.

[7] K.Yoshikawa et al., *28th DAC*, pp.112-117,1991.

[8] K.J.Singh and A.Sangiovanni-Vincentelli, *27th DAC*, pp.357-360,1990.

[9] S.Lin and M.Marek-Sadowska, *Proc. of the European Conference on Design Automation*, pp.539-544,1991.

[10] M.Murakata et al., *CICC-95*, pp.465-468,1995.

[11] H.B.Bakoglu, Circuits, Interconnections, and Packaging for VLSI, *Addison-Wesley*,1990.

[12] M.Igarashi et al., *SASIMI-93*,pp.253-244,1993.

[13] N.J.Nilsoon, Problem-Solving Methods in Artificial Intelligence, *McGraw-Hill*,1971.

[14] S.Prasitjutrakul and W.J.Kubitz, *IEEE Trans. on CAD*, vol.CAD-11,no.8,pp.1044-1051,1992.

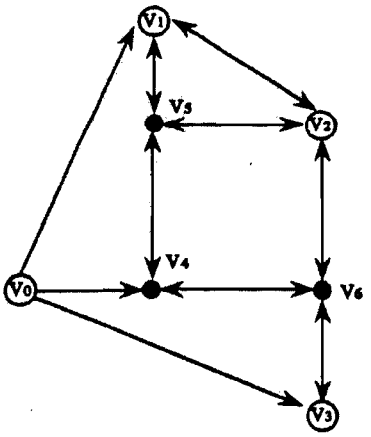


Figure 5: Example of directed planner graph. '→' are uni-directed edges and '↔' are bi-directed edges.

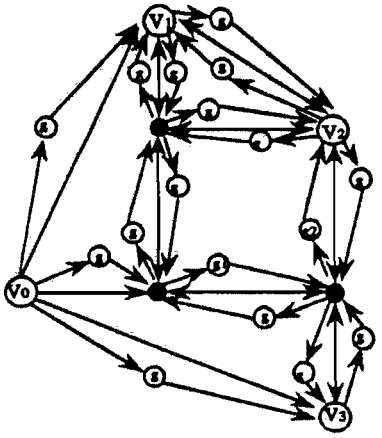


Figure 6: Example of fanout-graph. g, g_1, g_2 are inserted *buffer* .

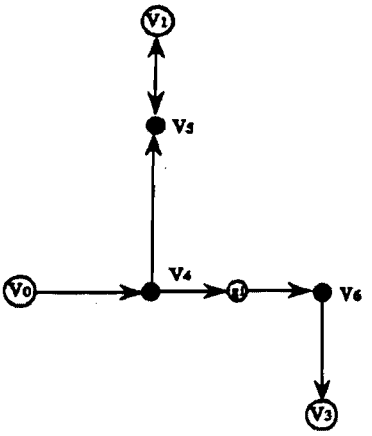


Figure 7: Example of start vertices

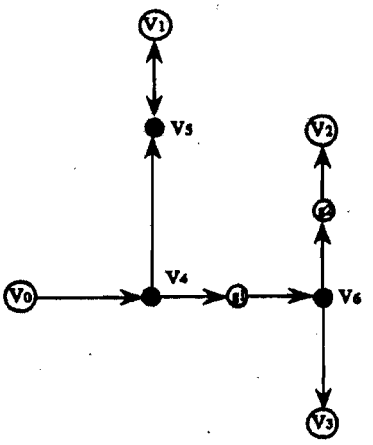


Figure 8: Example of maximized slack tree

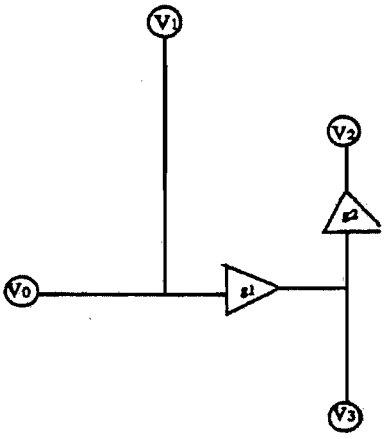


Figure 9: Example of fanout-tree configuration. g_1 and g_2 are inserted buffer cells.

TAB 36

Notice of Allowability	Application No.	Applicant(s)	
	09/579,966	SRINIVAS ET AL.	
	Examiner	Art Unit	
	Helen B Rossoshek	2825	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

All claims being allowable, PROSECUTION ON THE MERITS IS (OR REMAINS) CLOSED in this application. If not included herewith (or previously mailed), a Notice of Allowance (PTOL-85) or other appropriate communication will be mailed in due course. **THIS NOTICE OF ALLOWABILITY IS NOT A GRANT OF PATENT RIGHTS.** This application is subject to withdrawal from issue at the initiative of the Office or upon petition by the applicant. See 37 CFR 1.313 and MPEP 1308.

1. ☒ This communication is responsive to *amendments filed 06/28/2002*.
2. ☒ The allowed claim(s) is/are *1-8*.
3. ☒ The drawings filed on *05/26/2000* are accepted by the Examiner.
4. ☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).
   a) ☐ All b) ☐ Some* c) ☐ None of the:
      1. ☐ Certified copies of the priority documents have been received.
      2. ☐ Certified copies of the priority documents have been received in Application No. _____ .
      3. ☐ Copies of the certified copies of the priority documents have been received in this national stage application from the International Bureau (PCT Rule 17.2(a)).

   * Certified copies not received: _____.
5. ☐ Acknowledgment is made of a claim for domestic priority under 35 U.S.C. § 119(e) (to a provisional application).
   (a) ☐ The translation of the foreign language provisional application has been received.
6. ☐ Acknowledgment is made of a claim for domestic priority under 35 U.S.C. §§ 120 and/or 121.

Applicant has THREE MONTHS FROM THE "MAILING DATE" of this communication to file a reply complying with the requirements noted below. Failure to timely comply will result in ABANDONMENT of this application. THIS THREE-MONTH PERIOD IS NOT EXTENDABLE.

7. ☐ A SUBSTITUTE OATH OR DECLARATION must be submitted. Note the attached EXAMINER'S AMENDMENT or NOTICE OF INFORMAL PATENT APPLICATION (PTO-152) which gives reason(s) why the oath or declaration is deficient.

8. ☐ CORRECTED DRAWINGS must be submitted.
   (a) ☐ including changes required by the Notice of Draftsperson's Patent Drawing Review (PTO-948) attached
      1) ☐ hereto or 2) ☐ to Paper No. ____.
   (b) ☐ including changes required by the proposed drawing correction filed _____, which has been approved by the Examiner.
   (c) ☐ including changes required by the attached Examiner's Amendment / Comment or in the Office action of Paper No. _____.

Identifying indicia such as the application number (see 37 CFR 1.84(c)) should be written on the drawings in the top margin (not the back) of each sheet. The drawings should be filed as a separate paper with a transmittal letter addressed to the Official Draftsperson.

9. ☐ DEPOSIT OF and/or INFORMATION about the deposit of BIOLOGICAL MATERIAL must be submitted. Note the attached Examiner's comment regarding REQUIREMENT FOR THE DEPOSIT OF BIOLOGICAL MATERIAL.

Attachment(s)

1☐ Notice of References Cited (PTO-892)
2☐ Notice of Informal Patent Application (PTO-152)
3☐ Notice of Draftperson's Patent Drawing Review (PTO-948)
4☐ Interview Summary (PTO-413), Paper No.____ .
5☐ Information Disclosure Statements (PTO-1449), Paper No. ____.
6☐ Examiner's Amendment/Comment
7☐ Examiner's Comment Regarding Requirement for Deposit of Biological Material
8☒ Examiner's Statement of Reasons for Allowance
9☐ Other .

U.S. Patent and Trademark Office
PTO-37 (Rev. 04-01) Notice of Allowability Part of Paper No. 7 .

DETAILED ACTION

1. This office action is in response to application 09/579,966 filed 05/26/2000 and amendments filed 06/26/2002.

2. Claims 1-8 remain pending in the application.

Allowable Subject Matter

3. Claims 1-8 are allowed over prior art of record.

4. The examiner finds applicant's arguments persuasive. The following is an examiner's statement of reasons for allowance: the prior art of record does not teach a method of estimating the effect of adjacent wires on the capacitance of a signal wire in a circuit design which provides estimates based on an estimate of congestion in the design; subdividing the chip area of a circuit design to be placed and routed into a coarse grid of buckets (units with routing wires running through it in the vertical direction in one layer and routing wires running through it in the horizontal direction in another layer); an estimate of congestion in each bucker is calculated from an estimated amount of routing space available on the bucket and estimated consumption of routing resources (space) by a global router; this congestion score (ratio) is then used to determine the spacing of the wires in the bucket (modifying the congestion score) which is in turn used to estimate the capacitance of the wire segment in the bucket.

5. Any comments considered necessary by applicant must be submitted no later than the payment of the issue fee and, to avoid processing delays, should preferably accompany the issue fee. Such submission should be clearly labeled "Comments on Statement of Reasons for Allowance".

6. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Helen B Rossoshek whose telephone number is 703-305-3827. The examiner can normally be reached on 8:30-5:00.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Matthew S Smith can be reached on 703-308-1323. The fax phone numbers for the organization where this application or proceeding is assigned are 703-872-9318 for regular communications and 703-872-9319 for After Final communications.

Any inquiry of a general nature or relating to the status of this application or proceeding should be directed to the receptionist whose telephone number is 703-308-0956.

HR
September 5, 2002

VUTHE SIEK
PRIMARY EXAMINER

TAB 37

ALGORITHMS FOR VLSI PHYSICAL DESIGN AUTOMATION

THIRD EDITION

Distributors for North, Central and South America:
Kluwer Academic Publishers
101 Philip Drive
Assinippi Park
Norwell, Massachusetts 02061 USA
Telephone (781) 871-6600
Fax (781) 871-6528
E-Mail <kluwer@wkap.com>

Distributors for all other countries:
Kluwer Academic Publishers Group
Distribution Centre
Post Office Box 322
3300 AH Dordrecht, THE NETHERLANDS
Telephone 31 78 6392 392
Fax 31 78 6546 474
E-Mail <orderdept@wkap.nl>


Electronic Services <http://www.wkap.nl>

Library of Congress Cataloging-in-Publication Data

A C.I.P. Catalogue record for this book is available
from the Library of Congress.

Printed on acid-free paper.

Printed in the United States of America

nt problem. In
ng gestion is

layout based on

the constraints,

al design envi-
in [MTDL90].
cell layouts has
n has been dis-
cut and slicing
rmance in stan-
ssed in [SL90].
L88]. An ap-
been developed
monstrate that
me fault toler-
In [KK97], the
t its projected
ault-tolerance.
algorithm that
ed. In [MK98],
d modules has
v : oach for
every module
Simulating the
yout to decide
ulation of the

Chapter 7

Placement

Placement is a key step in physical design cycle. A poor placement consumes larger areas, and results in performance degradation. It generally leads to a difficult or sometimes impossible routing task. The input to the placement phase is a set of blocks, the number of terminals for each block and the netlist. If the layout of the circuit within a block has been completed then the dimensions of the block are also known. Placement phase is very crucial in overall physical design cycle. It is due to the fact, that an ill-placed layout cannot be improved by high quality routing. In other words, the overall quality of the layout, in terms of area and performance is mainly determined in the placement phase.

The placement of block occurs at three different levels.

1. **System level placement:** At system level, the placement problem is to place all the PCBs together so that the area occupied is minimum. At the same time, the heat generated by each of the PCBs should be dissipated properly so that the system does not malfunction due to overheating of some component.

2. **Board level placement:** At board level, all the chips on a board along with other solid state devices have to be placed within a fixed area of the PCB. All blocks are fixed and rectangular in shape. In addition, some blocks may be pre-placed. The PCB technology allows mounting of components on both sides. There is essentially no restriction on the number of routing layers in PCB. Therefore in general, the nets can always be routed irrespective of the quality of components placement. The objective of the board-level placement algorithms is twofold: minimization of the number of routing layers; and satisfaction of the system performance requirements. For high performance circuits, the critical nets should have lengths which are less than a specified value and hence the placement algorithms should place the critical components closer together. Another key placement problem is the temperature profile of the board. The heat dissipation on a PCB should be uniform, i.e., the chips which generate maximum heat should not be placed closer to each other. If MCMs are

7.2 Classification of Placement Algorithms

The placement algorithms can be classified on the basis of :

1. the input to the algorithms,
2. the nature of output generated by the algorithms, and
3. the process used by the algorithms.

Depending on the input, the placement algorithms can be classified into two major groups: *constructive placement* and *iterative improvement* methods. The input to the constructive placement algorithms consists of a set of blocks along with the netlist. The algorithm finds the locations of blocks. On the other hand iterative improvement algorithms start with an initial placement. These algorithms modify the initial placement in search of a better placement. These algorithms are typically used in an iterative manner until no improvement is possible.

The nature of output produced by an algorithm is another way of classifying the placement algorithms. Some algorithms generate the same solution when presented with the same problem, i.e., the solution produced is repeatable. These algorithms are called *deterministic* placement algorithms. Algorithms that function on the basis of fixed connectivity rules (or formulae) or determine the placement by solving simultaneous equations are deterministic and always produce the same result for a particular placement problem. Some algorithms, on the other hand, work by randomly examining configurations and may produce a different result each time they are presented with the same problem. Such algorithms are called as *probabilistic* placement algorithms.

The classification based on the process used by the placement algorithms is perhaps the best way of classifying these algorithms. There are two important class of algorithms under this classification: simulation based algorithms and partitioning based algorithms. Simulation based algorithms simulate some natural phenomenon while partitioning based algorithms use partitioning for generating the placement. The algorithms which use clustering and other approaches are classified under 'other' placement algorithms.

7.3 Simulation Based Placement Algorithms

There are many problems in the natural world which resemble placement and packaging problems. Molecules and atoms arrange themselves in crystals, such that these crystals have minimum size and no residual strain. Herds of animals move around, until each herd has enough space and it can maintain its predator-prey relationships with other animals in other herds. The simulation based placement algorithms simulate some of such natural processes or phenomena. There are three major algorithms in this class: simulated annealing, simulated evolution and force directed placement. The simulated annealing algorithm simulates the annealing process which is used to temper metals. Simulated

evolution simulates the biological process of evolution while the force directed placement simulates a system of bodies attached by springs. These algorithms are described in the following subsections.

7.3.1 Simulated Annealing

Simulated annealing is one of the most well developed placement methods available [BJ86, GS84, Gro87, Haj88, HRSV86, LD88, Oht86, RSV85, RVS84, SL87, SSV85]. The simulated annealing technique has been successfully used in many phases of VLSI physical design, e.g., circuit partitioning. The detailed description of the application of simulated annealing method to partitioning may be found in Chapter 5. Simulated annealing is used in placement as an iterative improvement algorithm. Given a placement configuration, a change to that configuration is made by moving a component or interchanging locations of two components. In case of the simple pairwise interchange algorithm, it is possible that a configuration achieved has a cost higher than that of the optimum but no interchange can cause a further cost reduction. In such a situation the algorithm is trapped at a local optimum and cannot proceed further. Actually this happens quite often when this algorithm is used on real life examples. Simulated annealing avoids getting stuck at a local optimum by occasionally accepting moves that result in a cost increase.

In simulated annealing, all moves that result in a decrease in cost are accepted. Moves that result in an increase in cost are accepted with a probability that decreases over the iterations. The analogy to the actual annealing process is heightened with the use of a parameter called *temperature* T. This parameter controls the probability of accepting moves which result in an increased cost. More of such moves are accepted at higher values of temperature than at lower values. The acceptance probability can be given by $e^{\frac{-\Delta C}{T}}$, where ΔC is the increase in cost. The algorithm starts with a very high value of temperature which gradually decreases so that moves that increase cost have lower probability of being accepted. Finally, the temperature reduces to a very low value which causes only moves that reduce cost to be accepted. In this way, the algorithm converges to a optimal or near optimal configuration.

In each stage, the configuration is shuffled randomly to get a new configuration. This random shuffling could be achieved by displacing a block to a random location, an interchange of two blocks, or any other move which can change the wire length. After the shuffle, the change in cost is evaluated. If there is a decrease in cost, the configuration is accepted, otherwise, the new configuration is accepted with a probability that depends on the temperature. The temperature is then lowered using some function which, for example, could be exponential in nature. The process is stopped when the temperature has dropped to a certain level. The outline of the simulated annealing algorithm is shown in Figure 7.4.

The parameters and functions used in a simulated annealing algorithm determine the quality of the placement produced. These parameters and functions include the cooling schedule consisting of initial temperature (*init_temp*),

Algorithm
begin
temp =
place =
while (
whi
tem
end.

Figu

final temperature
perature (SCHED
each temperature,
acceptance probat
these parameters
short time.

Sechen and S
a standard cell pl
TimberWolf is on
rithm, the parame
ule, *init_temp* =
where $\alpha(T)$ is a c
taken relatively l
just starts, whic
the medium rang
temperature char
is again taken 0.8
total of 117 temp
Figure 7.5. The
the circuit, e.g.,
for a 3000-cell ci
generated by ma

1. the displac
2. the interch
3. an orienta

3. Width and Height of a module for orientation optimization.

The initial sequence-pair is made as $S_1 = S_2$, which corresponds to a linear horizontal arrangement of modules. The temparature was decreased exponentially. Operation 1 was performed with higher probability in higher temperatures and operation 3 was performed with higher probability in lower temperatures to achieve better results.

The above technique can be extended to consider wire lengths also.

7.3.5 Comparison of Simulation Based Algorithms

Both the simulated annealing and simulated evolution are iterative and probabilistic methods. They can both produce optimal or near-optimal placements, and they are both computation intensive. However, the simulated evolution has an advantage over the simulated annealing by using the history of previous trial placements. The simulated annealing can only deal with one placement configuration at a time. In simulated annealing it is possible that a good configuration maybe obtained and then lost when a bad configuration is introduced later. On the other hand, the good configuration has much better chance to survive during each iteration in simulated evolution since there are more than one configurations being kept during each iteration. Any new configuration is generated by using several configurations in simulated evolution. Thus, history of previous placements can be used. However, the genetic method has to use much more storage space than the simulated annealing since it has to memorize all individual configurations in the population. Unlike simulated annealing and simulated evolution, force directed placement is applicable to general designs, such as full custom designs. The force-directed methods are relatively faster compared to the simulated annealing and genetic approaches, and can produce good placement.

7.4 Partitioning Based Placement Algorithms

This is an important class of algorithms in which the given circuit is repeatedly partitioned into two subcircuits. At the same time, at each level of partitioning, the available layout area is partitioned into horizontal and vertical subsections alternately. Each of the subcircuits so partitioned is assigned to a subsection. This process is carried out till each subcircuit consists of a single gate and has a unique place on the layout area. During partitioning, the number of nets that are cut by the partition is usually minimized. In this case, the group migration method can be used.

7.4.1 Breuer's Algorithm

The main idea for Breuer's algorithm [Bre77a, Bre77b] is to reduce the number of nets being cut when the circuit is partitioned. Various objective functions

have been develo
below.

1. **Total net-**
partitioning
by both ho
value is shc
length [Bre'

2. **Min-max**
and gate ar
are routed
is the chan
tive functic
the channe
number of
nels that h
channel.

3. **Sequentia**
introduced
two object
it is very
function r
each partit
is easier tc
nets cut.

In addition to t
placement proc

1. **Cut Orie**
the chip i
partitione
blocks for
line and t
procedure
yield goo
processin
to be par
block int
fit in the
than the

2. **Quadrat**
is partiti
vertical c
the parti

have been developed for this method. These objective functions are as given below.

1. **Total net-cut objective function:** All the nets that are cut by the partitioning are taken into account. This sum includes all nets cut by both horizontal and vertical partitioning cut lines. Minimizing this value is shown to be equivalent to minimizing the semi-perimeter wirelength [Bre77a, Bre77b].

2. **Min-max cut value objective function:** In the case of standard cells and gate arrays, the channel width depends on the number of nets that are routed through the channel. The more the number of nets the larger is the channel width and therefore the chip size. In this case the objective function is to reduce the number of nets cut by the cut line across the channel. This will reduce the congestion in channels having a large number of nets but at the expense of routing them through other channels that have a fewer number of nets or through the sparser areas of the channel.

3. **Sequential cut line objective function:** A third objective function is introduced to ease the computation of net cuts. Even though the above two objective functions represent a placement problem more accurately, it is very difficult to compute the minimum net cuts. This objective function reduces the number of nets cut in a sequential manner. After each partition, the number of nets cut is minimized. This greedy approach is easier to implement, however, it may not minimize the total number of nets cut.

In addition to the different objective functions, Breuer also presented several placement procedures in which different sequence of cut lines are used.

1. **Cut Oriented Min-Cut Placement:** Starting with the entire chip, the chip is first cut by a partition into two blocks. The circuit is also partitioned into two subcircuits so that the net cut is minimized. All the blocks formed by the partition are further partitioned by the second cut line and this process is carried out for all the cut lines. This partitioning procedure is sequential and easy to implement but it does not always yield good results because of the following two reasons. Firstly, while processing a cut line, the blocks created by the previous cut lines have to be partitioned simultaneously. Secondly, when a cut line partitions a block into two, the blocks to be placed in one of the partition might not fit in the partition created by the cut line as it might require more space than the block to be placed in the other partition (see Figure 7.9(a)).

2. **Quadrature Placement Procedure:** In this procedure, each region is partitioned into four regions of equal sizes by using horizontal and vertical cut lines alternatively. During each partitioning, the cutsize of the partition is minimized. As it cuts through the center and reduces the

TAB 38

AN EVOLUTIONARY STRATEGY FOR THE GLOBAL PLACEMENT OF MACRO CELLS

Chong-Min Kyung
Korea Advanced Institute of
Science and Technology,
Cheongryang, Seoul, Korea
P.O. Box 150, Phone: 966-1931 ext. 3722,

Peter V. Kraus and Dieter A. Mlynski
Institut für Theoretische
Elektrotechnik und Meßtechnik
Universität Karlsruhe (TH)
Kaiserstr. 12
D-7500 Karlsruhe, West Germany
Phone: FRG (721)608-2620

Abstract

This paper describes a new approach implemented as a program called EVAN, for the global placement of macro cells. EVAN consists of four major subprocedures, i.e., incremental growth, reorientation, recalculation of the net and module positions and diffusion of the circuit modules. Unlike previous approaches such as min-cut or FDR-based techniques where the shape, size and orientation of the circuit modules are considered too late thus severely disturbing the already obtained distribution, EVAN considers the circuit connectivity, size, shape and orientation of the modules simultaneously in an evolutionary fashion. In the benchmark data [10] examples, it is shown, that the amount of inter-module overlaps was significantly reduced making the remaining packing process a much easier task.

1 Introduction

Circuit placement in VLSI or PCB layout has been one of the most frequently addressed CAD problems during the last decade. Quite satisfactory results have been obtained for the placement of standard cells using min-cut [1], [2], force-directed relaxation (FDR) -based techniques [3], [4], [5], [6], [7] and simulated annealing [8], [9], due to the rather uniform size/shape characteristics in the standard cells. On the other hand, the problem of placing macro cells has never been so far successfully tackled due to the fact that the shapes/sizes of macro cells can be widely varying among themselves.

Such techniques as min-cut [1], [2], FDR [3], or some variation of these [4], [5], [6] which were quite successful in standard cells may not be so in macro cell placement. For example, in min-cut techniques only the size, but not the shape of the circuit modules are considered in the hierarchical bipartitioning of the chip space. Moreover, the circuit connectivity is not reflected into the module placement in a global sense, but in a greedy fashion. (It tries to minimize the wire cross-overs too aggressively.) In FDR-based techniques, the circuit connectivity is globally considered in obtaining the distribution of modules, but the sizes of the modules, unfortunately, are all considered to be zero. This fact is a serious disadvantage especially in the macro cell placement.

We can roughly divide the whole placement into global placement and packing (or detailed placement). The global placement accepts the circuit connectivity, shape/size of the modules, the pin/pad location and the chip space specifiaction as input and results in a relative placement of modules such that the estimated wire length and the chip area is minimal. In the conventional techniques such as min-cut and FDR however, such global distribution of macro cells as obtained by these techniques has to be severely perturbed in the following packing process, because the shape and size information of circuit modules which is especially important in macro cell placement has not been adequately reflected in these techniques.

Even the simulated annealing, not to mention its long computing time, has not been so far successful with macro cell placement. This is because such blind trial operations as 'exchange' and 'move' of the module positions generate significant overlaps with neighbouring modules or voids, which are very difficult to remove in 2-dimensional chip plane. (This becomes a 1-dimensional problem in the standard cell placement, which has been rather successfully treated using simulated annealing [9]).

In this paper, we describe an evolution strategy called EVAN for the global placement of macro cells. A salient feature of EVAN is that it smoothly adapts the initial dot distribution obtained by FDR technique to the final (nearly packed) placement result by considering such information as size/shape, orientation of modules, pin location, overlaps among modules and chip space constraint simultaneously and incrementally. The individual steps in EVAN, that is, incremental growth, rotation and diffusion of modules and recalculation of the net and module positions are explained in section 2. Some examples are given in section 3, and the conclusion follows.

2 Global Placement Strategy

Overall flowchart of the program EVAN for the global placement of macro cells is shown in Fig. 1. EVAN begins with an initial distribution of center-of-module positions obtained by FDR. The

CH2868-8/90/0000-1668$1.00 © 1990 IEEE

five subprocedures within the shaded box in Fig. 1 forms a loop in the whole EVAN program. We have chosen FDR as a means of providing the initial module positions, because the locations of modules are determined with a global view on the circuit connectivity; thus the only drawback of FDR that it does not consider the shape/size of the modules are gracefully compensated for by applying the following subprocedures as a loop in an incremental fashion. Each of the subprocedures is explained below.

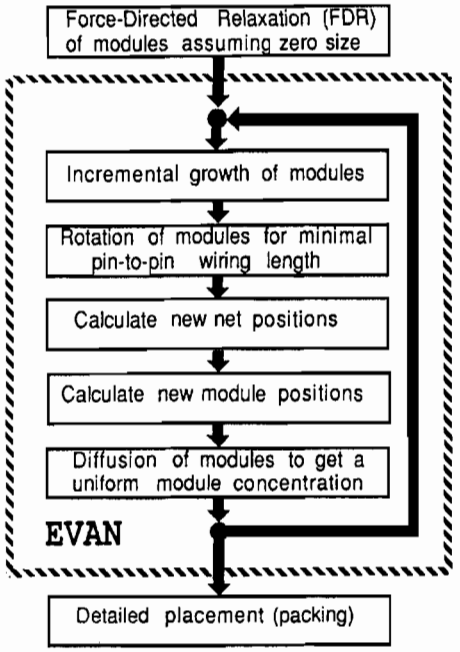


Figure 1: Global Placement Strategy

2.1 Incremental growth

The size of the modules are all assumed to be zero in the FDR process. Fig. 2 illustrates the process of incremental growth of a module to its final size. This is necessary to gracefully adapt the dot distribution pattern to the final module packing. This approach seems to be better than such techniques as FOCUP or min-cut placement, where the shape (aspect ratio) is suddenly reflected, thus possibly corrupting the earlier 'dot' distribution.

2.2 Rotation of the modules

In the same line of thought as above, the orientation of modules should be considered in an early stage of the whole placement process. All possible orientations of the modules including mirror reflection are regarded, such that the best module orientation is the one, where the sum of the half-perimeter of the minimum bounding bonds of the corresponding nets is minimized. Modules can only be rotated in 90 deg. steps.

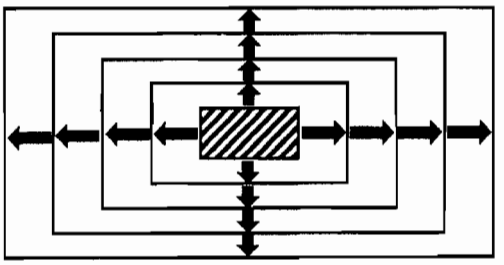


Figure 2: Incremental Growth. The innermost shaded rectangle represents the modules size in the first iteration, the outermost rectangle represents the final module size.

2.3 Calculation of the new net and module positions

After each rotation step, new net center positions are calculated assuming the module positions are fixed. Then, the new module positions are recalculated with the fixed net positions. This we do to prevent the system being trapped in a local minimum at an early stage of the process.

2.4 Diffusion of the modules

The diffusion is the process of relieving the overlaps among modules by letting the modules diffuse from overcrowded region into less-crowded region. It also guides the modules within the restricted chip region. Each diffusion cycle consists of 4 phases, each one representing the direction of 'wind', i.e., East, South, North and West. The whole chip is discretized with a grid as shown in Fig. 3. At each phase of a diffusion cycle, the behaviour of module i is represented by a move flag m_i, which equals 0 if it does not move, and equal 1 if it moves one grid step in the direction of the wind at that time. The resultant population of a grid cell (i,j) as a function of m's can be represented, for example in the East-phase, as follows:

$$A(i,j) = \sum_{k \in G_{i,j}} a_k(1-m_k) + \sum_{k \in G_{i-1,j}} a_k m_k$$

$G_{i,j}$: grid cell (i,j),
a_k: area of module k in cell (i,j),
m_k: move flag of module k.

The desired population of modules in grid cell (i,j) is represented as $B(i,j)$, which equals 1, if the grid cell is within an allowed placement region, and zero within the forbidden area, i.e.,

$$B(i,j) = \begin{cases} 1 & \text{within allowed area and} \\ 0 & \text{within forbidden area.} \end{cases}$$

The function, which has to be minimized is the squared difference of $A(i,j)$ and $B(i,j)$ over all the grid cells in the chip space, as given by,

$$X = \sum_{i,j} \big(A(i,j) - B(i,j)\big)^2.$$

The value of each m_k for $k = 1, \ldots, N$, (N: number of modules) can be obtained by solving a set of N linear equations,

$$\frac{\partial X}{\partial m_k} = 0, \qquad \text{for } k = 1, \ldots, N$$

If the resultant value of m_k is closer to 1, module k is moved, otherwise not.

3 Experimental Results

The EVAN algorithm is implemented as a C program running on VAX/VMS. In Fig. 4, the mechanism of the whole global placement strategy is illustrated. The 16 modules in this example are square and have all the same sizes. In Fig. 5, an example of the placement of 33 rectangular modules of a benchmark circuit called AMI33 [10] is illustrated.

References

[1] M.A. Breuer, "Min-cut placement", J. Design Automation and Fault Tolerant Computing, vol. 1, pp.343-382, Oct. 1977.

[2] A.E. Dunlop and B.W. Kernighan, "A procedure for placement of standard-cell VLSI circuits", IEEE Trans. on CAD of IC's and Systems, vol. CAD-4, pp. 92-98, Jan. 1985.

[3] N.R. Quinn and M.A. Breuer, "A force-directed component placement procedure for printed circuit boards", IEEE Transactions on Circuits and Systems, vol. CAS-26, pp. 377-388, 1979.

[4] R.S. Tsay, E.S. Kuh and C.P. Hsu, "Module placement for large chips based on sparse linear equations", Int. J. Circuit Theory and Applications, vol. 16, pp. 416-423, 1988.

[5] C.K. Cheng and E.S. Kuh, "Module placement based on resistive network optimization", IEEE Trans. on CAD of IC's and Systems, vol. CAD-3, pp. 218-225, July 1984.

[6] J.M. Kleinhans, G. Sigl and F.M. Johannes, "GORDIAN: A new global optimization/rectangle dissection method for cell placement", Proc. ICCAD, pp. 506-509, 1988.

[7] G.J. Wipfler, M. Wiesel and D.A. Mlynski, "A combined force and cut algorithm for hierarchical VLSI layout", Proc. 19th DAC, pp. 671-676, 1982.

[8] S. Kirkpatrick, C.D. Gelatt and M.P. Vecchi, "Optimization by simulated annealing", Science, vol 220, pp. 671-680, May 1983.

[9] C. Sechen and A. Sangiovanni-Vincentelli, "TimberWolf 3.2: A new standard cell placement and global routing package", Proc. 23rd DAC, pp. 432-439, June 1986.

[10] B. Preas, "Benchmarks for cell-based layout systems", Proc. 24th DAC, pp. 319-320, 1987.

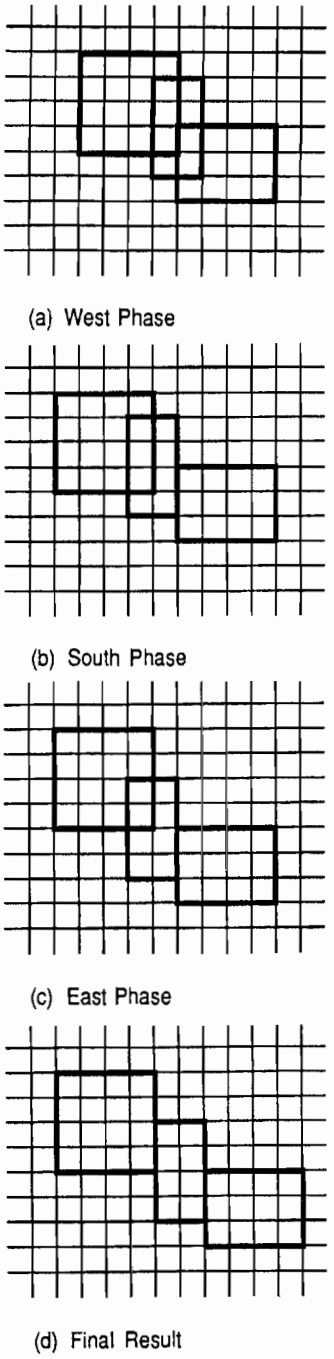


Figure 3: Diffusion of modules. Initial position is shown in (a). After W-phase, (b) is obtained, which becomes (c) after S-phase and then finally to (d) after E-phase.

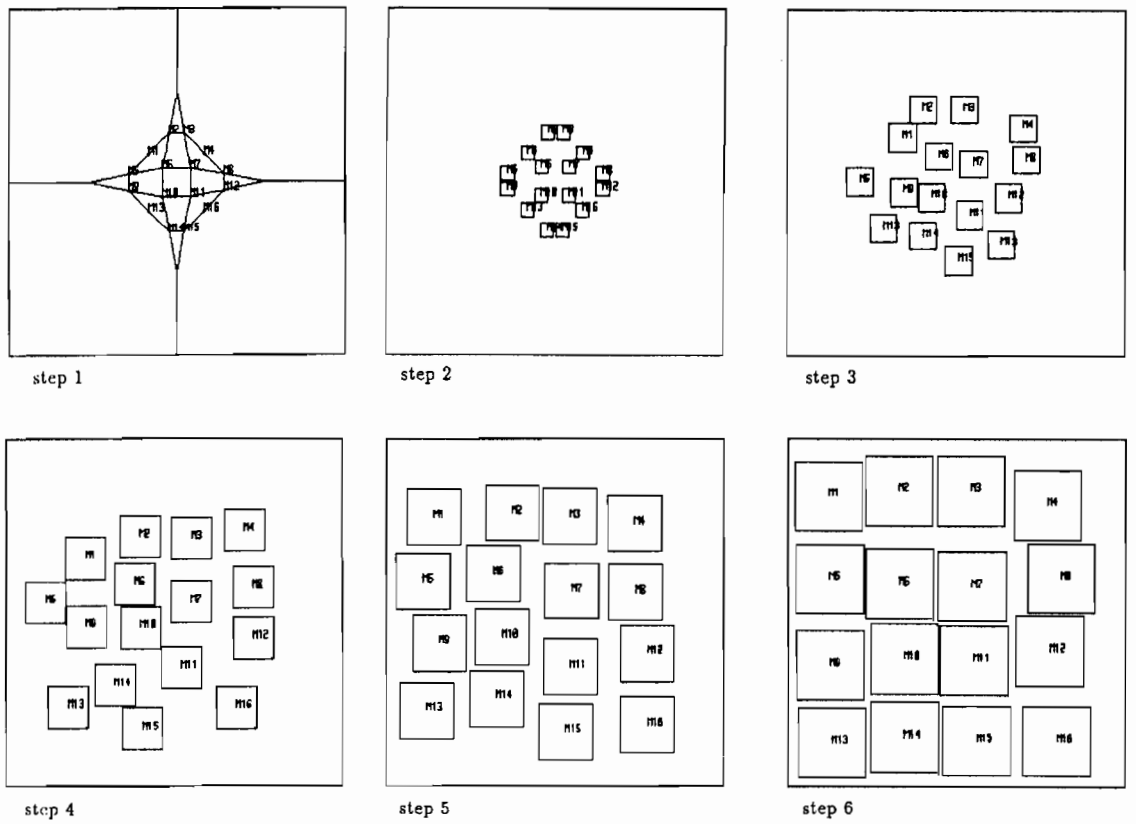


Figure 4: Example of 16 modules connected in a 2-dim. mesh pattern showing the intermediate steps in the EVAN procedure

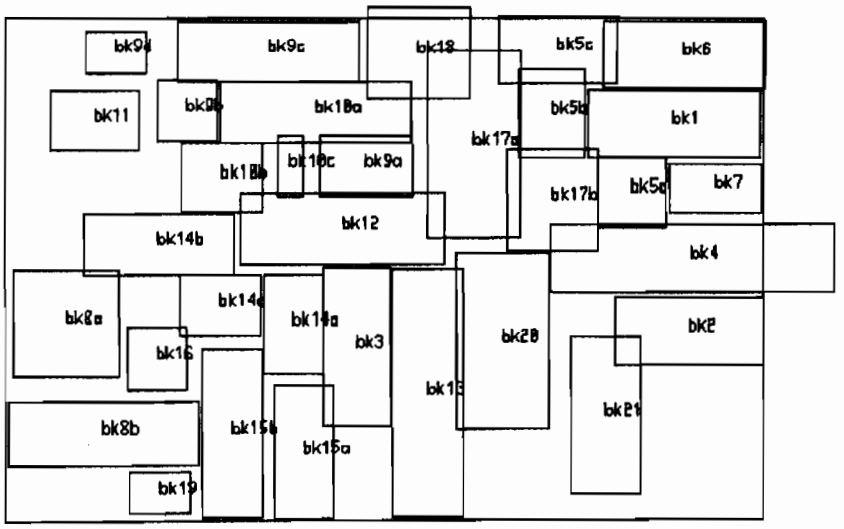


Figure 5: Example ami33

1671

TAB 39

Shuji Tsukiyama (S'76–M'77) was born in Osaka, Japan, on November 23, 1949. He received the B.S., M.S., and Ph.D. degrees all in electronic engineering from the Faculty of Engineering, Osaka University, Suita, Osaka, in 1972, 1974, and 1977, respectively.

He has been with the Department of Electronic Engineering, Osaka University, as a Research Assistant. During the academic year of 1978–1979, he was with the Electronics Research Laboratory, University of California, Berkeley, as a Visiting Research Engineer. His main interest is in graph theory, computational complexity, and computer-aided design.

Dr. Tsukiyama is a member of the Institute of Electronic and Communication Engineers of Japan.

Isao Shirakawa (S'62–M'68) was born in Toyama Prefecture, Japan, on September 12, 1939. He received the B.S., M.S., and Ph.D. degrees, all in electronic engineering from the Faculty of Engineering, Osaka University, Suita, Osaka, Japan, in 1963, 1965, and 1968, respectively.

Since 1968 he has been with the Department of Electronic Engineering at the same university, where he is now an Associate Professor. During the academic year of 1974–1975 he was with the Electronics Research Laboratory, University of California, Berkeley, as a Visiting Research Engineer. His current interest is mainly in applied graph theory, design automation, and computer-aided design associated with layout problems.

Dr. Shirakawa is a member of the Institute of Electronic and Communication Engineers of Japan and Information Processing Society of Japan.

Shigeo Asahara was born in Osaka, Japan, on September 17, 1953. He received the B.S. and M.S. degrees in electronic engineering from Osaka University, Osaka, Japan, in 1976 and 1978, respectively.

He is currently a Graduate Student working towards the Ph.D. degree at the same university. His main interest is in layout for PWB's and LSI's.

A Forced Directed Component Placement Procedure for Printed Circuit Boards

NEIL R. QUINN, JR., AND MELVIN A. BREUER, SENIOR MEMBER, IEEE

Abstract—**This paper deals with the problem of placing components on a carrier, such as a printed circuit board (PCB). We present a new mathematical formulation of the concept of force directed placement, and describe an efficient computational procedure for solving the resulting system of equations. The placement procedure is broken down into two phases, Phase I being the "relative location phase," and Phase II being the "slot assignment or component overlap resolution phase." In Phase I of the procedure, we solve a set of simultaneous equations, based upon the interconnection topology of the system of components, in an endeavor to determine the optimum relative location of every component with respect to every other component. The equations are set up such that there are attractive forces between components sharing a common signal, and repulsive forces between components having no signals in common. The results of Phase I are often unacceptable from a physical standpoint because there is a great deal of overlap among the components. Phase II eliminates component overlap by either of two methods, depending upon the physical properties of the carrier. If the carrier is subdivided into slots, then the components are assigned to these slots using a criteria which minimizes the total distance that all components need be moved. We perform this assignment by using the linear assignment algorithm. If the carrier is such that components are allowed to reside anywhere, then a different technique to resolve component overlap is used.**

A parametric analysis of the procedure is given based upon 12 different PCB's. These results show comparisons of this method to the work of others, and provide some insight into the method's absolute merits.

Keywords: **Assignment, placement, layout, printed circuit boards, force-directed placement.**

I. The Placement Problem

THIS PAPER deals with the problem of placing components on a carrier, such as a printed circuit board (PCB) or ceramic substrate. Hanan and Kurtzberg [5] state, "in the real world, there are actually a number of

Manuscript received April 20, 1978. This work was supported in part by the National Science Foundation under Grant ENG74-18647.

N. R. Quinn, Jr., is with Environmental Management Systems, Inc., San Diego, CA.

M. A. Breuer is with the University of Southern California, Los Angeles, CA.

0098-4094/79/0600-0377$00.75

(possibly conflicting) goals which must be satisfied in a placement configuration: wire buildup in the routing channels must be kept within tolerable bounds; signal cross-talk must be eliminated without undue use of expensive shielding techniques; signal echoes must be eliminated; heat dissipation levels must be perserved, etc."

It is clearly impractical, if not impossible, to directly incorporate into a placement algorithm all of the real world goals mentioned above. However, minimizing the total weighted wire length of the circuit is one criteria that can be used to partially satisfy these goals en masse, and is the usual measure adopted for placement algorithms. We shall also use this objective.

A. Previous Placement Techniques

There are basically two different categories of placement problems, referred to as the continuous and noncontinuous (array) problems. For the *continuous case*, the placement media, such as a PCB, can be treated as a continuous plane on which the components to be placed are free to reside. In the *noncontinuous case*, the media is partitioned into an array of slots into which the components are placed (assigned).

A fairly complete discussion of the noncontinuous case can be found in Breuer [7]. The major contributors to the continuous case are Crocker, McGriffin, Naylor, and Vosper [3]; Scanlon [2]; and Fisk, Caskey, and West [4]. Each of these papers describes a different variation of a force directed placement procedure. Each of these techniques suffers from some technical obstacle which thus limits its effectiveness as a truly useful placement algorithm. It is the objective of this paper to present a refined technique which eliminates these obstacles.

The placement procedure presented here is broken down into two phases, Phase I being the "relative location phase," and Phase II being the "slot assignment or component overlap resolution phase."

II. The Relative Location Problem

We shall define the *relative location problem* as being the task of determining the correct relative position of every component with respect to every other component such that, with a minimum distortion of these positions by the procedures of Phase II, the signal nets of these components can be interconnected using a near minimal amount of wire.

To solve the relative location problem we shall develop a set of force equations governed by the interconnection topology (signal nets) of the system of components, the solution of which produces the desired result. The equations will be based upon the concept of mass as well as the forces of attraction and repulsion between the components. These equations will be formulated with the following characteristics in mind:

1) to behave similarly to the quadratic assignment problem by taking into account the positions of all the components;
2) to tend to minimize the minimum spanning tree (MST) [1] length needed to interconnect the signal nets;
3) to take into account the positions of fixed components such as connectors;
4) to be solvable in an efficient manner.

It would be ideal to have a fifth characteristic which states that "the positions of the slots into which components must be placed would be taken into account;" however, this is impractical from a computational standpoint, since it is analogous to the quadratic assignment problem which requires a solution time that is factorially bounded.

In developing the solution being presented in this paper, numerous concepts were evaluated. Some of these concepts worked well, others did not. A thorough presentation of both the concepts that did and did not work is presented in Quinn [14]. In this paper we will only present those concepts which proved fruitful.

A. Hooke's Law and the Placement of Components

Hooke's Law states that if two particles (components) are connected to each other by a spring (signal net), then there is an attractive force between these two particles (components) that is equal to the spring constant (function of the number of signal names that two components share in common) times the distance between the two particles (components).

Let (x_i,y_i) be the coordinates of component i, and let $\Delta x_{ij}=x_j-x_i$ and $\Delta y_{ij}=y_j-y_i$. Let $K_{ij}=K_{ji}$ be a constant proportional to the number of signals that components i and j have in common;

$$\overline{\Delta S_{ij}}=\begin{pmatrix}\Delta x_{ij}\\ \Delta y_{ij}\end{pmatrix}$$

be the vector distance between components i and j ($\overline{\Delta S_{ij}}=-\overline{\Delta S_{ji}}$), and let $\bar{F}_{ij}$ be the force exerted on component i by component j. Then $\overline{F_{ij}}=-K_{ij}\overline{\Delta S_{ij}}$. The magnitude of $\overline{\Delta S}$ will be denoted by ΔS, and as will be shown later, there are several analytical and computational advantages to choosing as our metric $\Delta S=|\Delta x|+|\Delta y|$ rather than the more conventional Euclidian distance $\Delta S=(\Delta x^2+\Delta y^2)^{1/2}$.

The x and y coordinate components of $\bar{F}_{ij}$, denoted by Fx_{ij} and Fy_{ij}, are given by the expressions

$$Fx_{ij}=-\bar{F}_{ij}\cdot\Delta x_{ij}/\Delta S_{ij}=-K_{ij}\cdot\Delta x_{ij}$$

and

$$Fy_{ij}=-K_{ij}\cdot\Delta y_{ij}.$$

If we assume M moveable components, then the total force on each component in the x and y direction is given by the equations

$$Fx_i=-\sum_{j=1}^{M}K_{ij}\Delta x_{ij}$$

and

$$Fy_i=-\sum_{j=1}^{M}K_{ij}\Delta y_{ij},\qquad \text{for } i=1,2,\cdots,M. \tag{1}$$

A solution to (1) consists in finding a set of values for all the x_i's and y_i's such that all the $Fx_i = Fy_i = 0$. One can quickly see that if all the components are free to move, then the solution to (1) is $x_1 = x_2 = x_3 = \cdots = x_M$ and $y_1 = y_2 = y_3 = \cdots = y_M$, i.e., all components collapse to a single point. This is due to the fact that there are no external forces acting on the system of components to keep them spatially separated.

This set of equations represents the basis for all force directed placement techniques [2]–[4]. The equations give some intelligible results in cases where there are three or more fixed components that have different x and y locations. Crocker *et al.* [3] use this method exclusively, since they are placing components in a package that has four connectors surrounding the four sides of the board. They solve the set of equations algebraically. In most cases, where the board has only one edge connector, this formulation is unacceptable. Hall [15] uses (1) with one additional constraint, i.e., if $\bar{X} = (x_1, x_2, \cdots, x_M)$ is a column vector and $\bar{X}^T$ its transpose, then Hall sets $\bar{X}\bar{X}^T = 1$, and similarly for $\bar{Y}$. This constraint keeps the components spatially separated.

B. Hooke's Law with Repulsion

To keep components spatially separated we have chosen to use a force of repulsion between unconnected components. This type of force is useful since it tends to spread out the components across the board as well as increase the probability of the components taking their correct relative positions.

There are two basic forms that a repulsion term can take, one which is inversely proportional to distance, or one which is constant, i.e., not dependent on distance. Experimental evidence indicates that either type of repulsion gives approximately the same results, but that a solution obtained using a constant repulsion term requires significantly less computer time.

Our formulation of the placement problem, using constant repulsion becomes

$$\bar{F}_{ij} = -K_{ij}\overline{\Delta S_{ij}} + \overline{R_{ij}} = -K_{ij}\begin{pmatrix}\Delta x_{ij}\\ \Delta y_{ij}\end{pmatrix} + \frac{R_{ij}}{\Delta S_{ij}}\begin{pmatrix}\Delta x_{ij}\\ \Delta y_{ij}\end{pmatrix}$$

where

$$R_{ij} = \begin{cases} 0, & \text{for } K_{ij} \neq 0 \\ 0, & \text{for } i = j \\ R, & \text{for } K_{ij} = 0 \end{cases}.$$

$\overline{R_{ij}}$ is a vector whose direction is that of $\overline{\Delta S_{ij}}$ and whose magnitude is R_{ij}. R is defined by the equation

$$R = \left(\frac{1}{C_R}\right)\cdot\left[\sum_{i=1}^{M}\sum_{j=1}^{M} K_{ij}\right]/T$$

where T equals the number of K_{ij}'s equal to 0. An explanation of R will be given shortly.

$\bar{F}_{ij}$ can be broken down into the components

$$Fx_{ij} = F_{ij}\cdot\Delta x_{ij}/\Delta S_{ij} = -K_{ij}\Delta x_{ij} + R_{ij}\Delta x_{ij}/\Delta S_{ij}$$

and

$$Fy_{ij} = F_{ij}\cdot\Delta y_{ij}/\Delta S_{ij} = -K_{ij}\Delta y_{ij} + R_{ij}\Delta y_{ij}/\Delta S_{ij}. \quad (2a)$$

Notice that the repulsion terms for Fx_{ij} and Fy_{ij} contain a scale factor equal to the cosine of the angle that the line connecting the two components makes with the x and y axes, respectively.

Expanding this formulation to cover all M moveable components, we obtain

$$Fx_i = \sum_{j=1}^{M} Fx_{ij} \quad \text{and} \quad Fy_i = \sum_{j=1}^{M} Fy_{ij}. \quad (2b)$$

These two repulsion equations contain the term ΔS_{ij}. This has a slightly undesirable side effect in that it mathematically couples these equations. This coupling effect makes the solution of these equations more difficult because there are now twice as many simultaneous equations.

C. Distance Measure

For both analytic and computational reasons we have chosen to define ΔS_{ij} by the Manhattan distance $|\Delta x_{ij}| + |\Delta y_{ij}|$. Computationally, it is preferred because ΔS_{ij} must be evaluated $M^2/2$ times for each iteration of our algorithm in order to determine the distance from every component to every other component. If the function $(\Delta x_{ij}^2 + \Delta y_{ij}^2)^{1/2}$ were used it would add significantly to the computational time because of the square root function. Analytically it is preferred because the sum of the contribution in the x and y directions equals 1, i.e.,

$$\Delta S = |\Delta x/(|\Delta x| + |\Delta y|)| + |\Delta y/(|\Delta x| + |\Delta y|)| = 1$$

for all values of Δx and Δy. If the Euclidian distance measure is used, then

$$\Delta S = |\Delta x/(\Delta x^2 + \Delta y^2)^{1/2}| + |\Delta y/(\Delta x^2 + \Delta y^2)^{1/2}|.$$

Now $1 \leq \Delta S < \sqrt{2}$, where $\Delta S = 1$ for Δx or $\Delta y = 0$, and $\Delta S = \sqrt{2}$ for $\Delta x = \Delta y \neq 0$. For this measure we get $\sqrt{2}$ greater repulsion along the diagonals than along the axes. This effect is undesirable, and, therefore, we use $\Delta S_{ij} = |\Delta x_{ij}| + |\Delta y_{ij}|$ throughout the remainder of this work. Other researchers [2], [4] have consistently used the Euclidian distance measure.

D. Other Types of Components

There are basically three types of components, namely moveable, fixed, and semimoveable. So far we have limited our discussion to the moveable type components. We will now turn our attention to the other types of components.

Fixed Components: Fixed components, by definition, are rendered immoveable throughout the placement process. These components are kept fixed by setting their positions (x_i, y_i) equal to a constant throughout the placement process. The way this is accomplished in the set of equations is by a simple indexing change on the summation limit. If there are N components total (both fixed and moveable), and M moveable components, then our equations become

$$Fx_i = \sum_{j=1}^{N} -K_{ij}\Delta x_{ij} + R_{ij}\Delta x_{ij}/\Delta S_{ij} \quad (3)$$

$$Fy_i = \sum_{j=1}^{N} -K_{ij}\Delta y_{ij} + R_{ij}\Delta y_{ij}/\Delta S_{ij} \quad (4)$$

for all $i = 1,2,\cdots,M$ moveable components.

The fixed components are generally groups of connector pins which have been preassigned signal names. However, there are many situations when the technology allows for the assignment of the signal names to the connector pins after the placement process. In these cases it is advisable to treat all the groups of connector pins along one edge of the board as a "bar" type connector. The bar type connector attracts components in one direction only. For example, if there is a "bar" type connector on the bottom edge of the board, all components connected to it will be attracted in the downward direction only. To implement this type of connector we need only make a small indexing change to the upper limit of the summation in one dimension.

If there are N components total, M moveable components, and B bar type components (assume the bar components exert a force in the y direction only) then our equations become

$$Fx_i = \sum_{j=1}^{N-B} -K_{ij}\Delta x_{ij} + R_{ij}\Delta x_{ij}/\Delta S_{ij}$$

and

$$Fy_i = \sum_{j=1}^{N} -K_{ij}\Delta y_{ij} + R_{ij}\Delta x_{ij}/\Delta S_{ij} \quad (5)$$

for $i = 1,2,\cdots,M$.

Semimoveable Components: A semimoveable component is free to move along one axis only. This constraint is easily implemented by keeping the component's position in the restricted direction constant. This again requires only a summation limit change in one dimension. If there are N components total, M moveable components, L semimoveable components (assume semimoveable components are free to move in the x direction only) then our equations become

$$Fx_k = \sum_{j=1}^{N} -K_{kj}\Delta x_{kj} + R_{kj}\Delta x_{kj}/\Delta S_{kj}$$

$$Fy_i = \sum_{j=1}^{N} -K_{ij}\Delta y_{ij} + R_{ij}\Delta y_{ij}/\Delta S_{ij} \quad (6)$$

for $i = 1,2,\cdots,M$ and $k = 1,2,\cdots,M+L$, where components $k+1, k+2,\cdots,M+L$ are restricted to move in the x direction only.

E. Effects on Solution by the Fixed and Semimoveable Components

If one solves the system of equations presented so far they would find that it leads to unacceptable results because in some cases the moveable components tend to surround the connectors. This has an adverse effect on the results of Phase II, because the components are not in their correct relative positions for assignment to slots on the board. To help alleviate this problem, we have found it useful to keep the bulk of the components at some fixed location on the board. To accomplish this we recall that to maintain the center of mass of a system of particles at a fixed location it is required to negate the net effect on any external forces acting on the system. In our case, the system of particles is the set of moveable components, and the external forces are those forces exerted on the moveable components by the fixed components. We can compute this external force, which we call the force on the

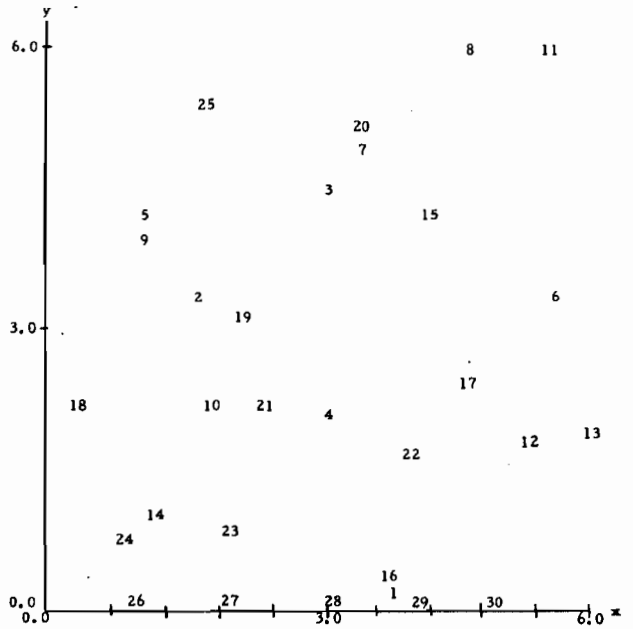


Fig. 1. (IL25) random initial placement of moveable components.

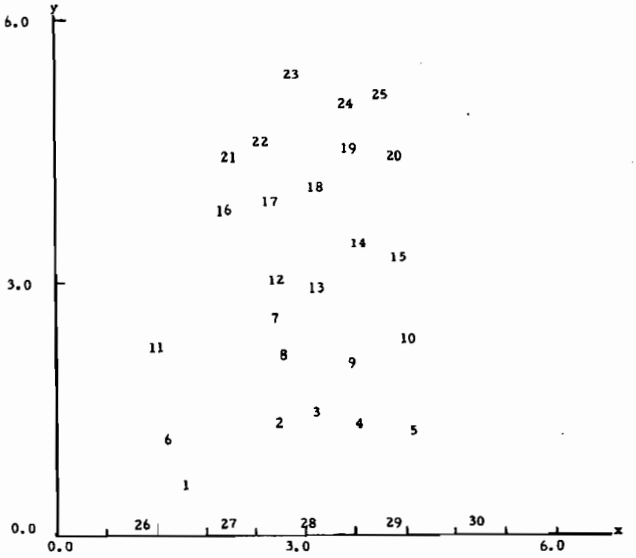


Fig. 2. (IL25) result of Phase I using (8).

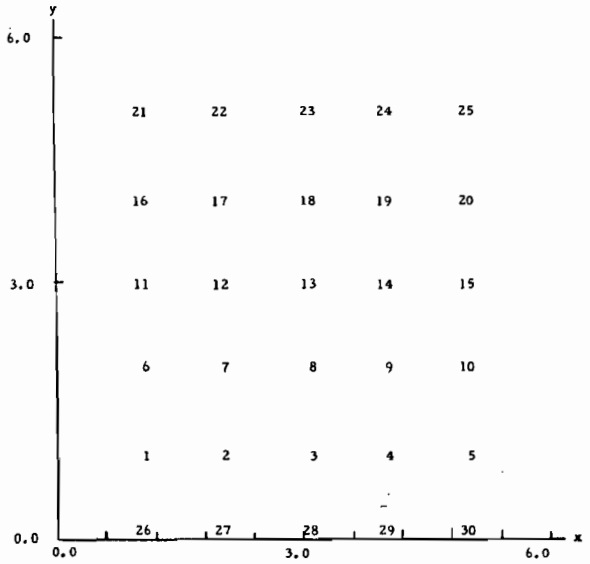


Fig. 3. (IL25) result from Phase II (optimum configuration for components on a 5×6 array board).

center of mass, denoted by *FCMX* and *FCMY*, by the following equations:

$$FCMX = \sum_{i=1}^{M} \sum_{j=M+1}^{N} -K_{ij}\Delta x_{ij}$$

$$FCMY = \sum_{i=1}^{M} \sum_{j=M+1}^{N} -K_{ij}\Delta y_{ij}. \tag{7}$$

Note, however, that

$$FCMX = \sum_{i=1}^{M} Fx_i \quad \text{and} \quad FCMY = \sum_{i=1}^{M} Fy_i$$

where Fx_i and Fy_i are defined in (2b) because the sum of all the forces acting on the moveable components must equal the total external force acting on these components, i.e., the total force between the moveable components and the fixed components.

In order to hold the center of mass (*CM*) of the moveable components fixed we need only to subtract a portion of *FCMX* and *FCMY* from each moveable component. Since there are M moveable components, we subtract from each component $FCMX/M$ and $FCMY/M$, and our final set of force equations becomes

$$Fx_i = \left[\sum_{j=1}^{N} -K_{ij}\Delta x_{ij} + R_{ij}\Delta x_{ij}/\Delta S_{ij}\right] - FCMX/M$$

$$Fy_i = \left[\sum_{j=1}^{N} -K_{ij}\Delta y_{ij} + R_{ij}\Delta y_{ij}/\Delta S_{ij}\right] - FCMY/M \tag{8}$$

for all $i = 1,2,\cdots,M$.

The initial position of the *CM* has a pronounced effect on the results. Its effect is covered in Section V. To the best of our knowledge, this problem of correctly aligning the moveable components with respect to the fixed components has not been previously addressed.

We shall illustrate our technique for obtaining a placement on our test board IL25.[1] This problem was designed so that the optimal placement is known. Components 1–25 are moveable and 26–30 are fixed. This problem had 65 signal nets and 122 edges in the minimal spanning tree. Figs. 1–3 show, respectively, the initial position of the components, the results of Phase I (using (8)), and the result of Phase II positioning upon a 5×6 array board using the results of Phase 1. This final solution is an optimal one.

III. Solution of Force Equations

In this section we will discuss an efficient method for solving the set of equations derived in Section II. Our goal is to develop a procedure that produces a solution at a reasonable cost (CPU time and storage), gives results that are in general comparable to or superior to those of other well-known techniques (Pairwise Interchange, Steinberg–Rutman, Force Directed Pairwise Relaxation, etc.), and obtains results for a variety of types of carriers. We now present our solution technique.

A. Solution Technique

We know *a priori* that the desired solution has the property that the forces on all components are equal to zero. By setting each of the forces in (8) equal to zero, we obtain the following set of algebraic equations:

$$\begin{aligned}
Fx_1 &= \left[\sum_{j=1}^{N} -K_{1j}\Delta x_{1j} + R_{1j}\Delta x_{1j}/\Delta S_{1j}\right] - FCMX/M = 0 \\
&\vdots \\
Fx_M &= \left[\sum_{j=1}^{N} -K_{Mj}\Delta x_{Mj} + R_{Mj}\Delta x_{Mj}/\Delta S_{Mj}\right] - FCMX/M = 0 \\
Fy_1 &= \left[\sum_{j=1}^{N} -K_{1j}\Delta y_{1j} + R_{1j}\Delta y_{1j}/\Delta S_{1j}\right] - FCMY/M = 0 \\
&\vdots \\
Fy_M &= \left[\sum_{j=1}^{N} -K_{Mj}\Delta y_{Mj} + R_{Mj}\Delta y_{Mj}/\Delta S_{Mj}\right] - FCMY/M = 0.
\end{aligned} \tag{9}$$

Since these equations are nonlinear, their solution is somewhat difficult to obtain. We have employed a modified version of the Newton–Raphson (NR) [6] technique for solving this system of equations. The NR method is very costly to use in terms of storage requirements and CPU time per iteration. In general, for the $2M$ simultaneous equations in $2M$ unknowns, we must calculate the Jacobian ($4M^2$ partial derivatives), and must then solve for the $2M$ increments p_i, q_i ($i = 1,2,\cdots,M$) by solving a set of $2M$ simultaneous linear equations in $2M$ unknowns. We shall present an easier method which, while perhaps not converging in as few a number of iterations as the NR method, is much easier to apply, and requires less com-

[1] A test board labeled ILn has n components.

putation time. Our method is a slightly modified version of the modified Newton–Raphson method for simultaneous equations (MNR) [6].

The MNR method consists of applying the one-dimensional NR method $2M$ times, once for each variable. Each time it is applied to a variable, the other variables remain fixed at their most current value.

Using this technique the computation time per iteration grows approximately as M^2, since for each iteration we must evaluate each function and its derivative using the newly updated value of the variables. If we make a slight modification to the MNR method by not using the updated values of the variables during an iteration, but instead update the values of the variables at the end of each iteration, then our computation time per iteration grows approximately as $M^2/2$. This reduction is significant for large M. The reason for the reduction in computation is that we can take into account certain geometrical symmetries that exist between the components if we know that they remain fixed during an iteration. The geometrical symmetries exist because the distances between components remains the same throughout each iteration.

Another reason for updating the values of the variables at the end of an iteration is that the *FCMX* and *FCMY* forces can be calculated quite easily if the components remain fixed during an iteration.

Applying this modified version of the NR method to (8), we obtain, for $i=1,2,\cdots,M$,

$$\frac{\partial Fx_i}{\partial x_i} = F'x_i = \sum_{j=1}^{N} -K_{ij} + R_{ij}\frac{|\Delta y_{ij}|}{\Delta S_{ij}^2} + \frac{1}{M}\sum_{j=M+1}^{N} K_{ij}$$

and

$$\frac{\partial Fy_i}{\partial y_i} = F'y_i = \sum_{j=1}^{N} -K_{ij} + R_{ij}\frac{|\Delta x_{ij}|}{\Delta S_{ij}^2} + \frac{1}{M}\sum_{j=M+1}^{N} K_{ij}$$

$$x_i(\text{new}) = x_i - \tfrac{1}{2}Fx_i/F'x_i$$

$$y_i(\text{new}) = y_i - \tfrac{1}{2}Fy_i/F'y_i. \qquad (10)$$

The factor of $1/2$ is due to the fact that any two components are attracted to each other by the same force, therefore we need move each component only $1/2$ the designated distance.

Our solution is derived as follows.

Solution Procedure: Method for solving the force equations.

Step 0: Preliminaries: for $i=1,2,\cdots,N$, set x_i and y_i to their initial values.

Step 1: For $i=1,2,\cdots,M$, set

$$Fx_i = \sum_{j=1}^{N} -K_{ij}\Delta x_{ij} + R_{ij}\Delta x_{ij}/\Delta S_{ij}$$

and

$$Fy_i = \sum_{j=1}^{N} -K_{ij}\Delta y_{ij} + R_{ij}\Delta y_{ij}/\Delta S_{ij}.$$

Step 2: For $i=1,2,\cdots,M$, set

$$F'x_i = \sum_{j=1}^{N} -K_{ij} + R_{ij}|\Delta y_{ij}|/\Delta S_{ij}^2 + \frac{1}{M}\sum_{j=M+1}^{N} K_{ij}$$

and

$$F'y_i = \sum_{j=1}^{N} -K_{ij} + R_{ij}|\Delta x_{ij}|/\Delta S_{ij}^2 + \frac{1}{M}\sum_{j=M+1}^{N} K_{ij}.$$

Step 3: Set

$$FCMX = \sum_{i=1}^{M} Fx_i \quad \text{and} \quad FCMY = \sum_{i=1}^{M} Fy_i.$$

Step 4: For $i=1,2,\cdots,M$, set

$$Fx_i = Fx_i - FCMX/M \quad \text{and} \quad Fy_i = Fy_i - FCMY/M.$$

Step 5: For $i=1,2,\cdots,M$, set

$$x_i = x_i - 1/2\cdot Fx_i/F'x_i \quad \text{and} \quad y_i = y_i - 1/2\cdot Fy_i/F'y_i.$$

Step 6: Set

$$\text{NORM} = \sum_{i=1}^{M} [|Fx_i| + |Fy_i|].$$

IF (NORM $>\epsilon$) GO TO Step 1.

Step 7: PRINT RESULTS.

B. Storage Requirements

This method requires $N(N-1)/2$ cells of storage for the $K=[K_{ij}]$ and $R=[R_{ij}]$ matrices, and $8M$ cells to store the vectors $\overline{Fx}$, $\overline{Fy}$, $\overline{F'x}$, $\overline{F'y}$, $\overline{Fx^{old}}$, $\overline{Fy^{old}}$, $\bar{x}$, $\bar{y}$.

C. Computational Complexity

We will describe the computational complexity per iteration of this method in terms of the number of arithmetic operations required. To do this we will take into account various symmetries that exist, and make some assumptions about the density of the K matrix. The reason for taking into account the symmetries of the problem is self explanatory. The reason for needing to assume the density of the K matrix is that the K and R matrices are mutually exclusive; therefore, the terms involving R need only be evaluated when the K elements are zero, and vice versa.

Table I gives the number of arithmetic operations per step for our solution procedure. For this table we will assume a 10-percent dense K matrix. A CDC Cyber 172 computer was used for all problems in this work and all times are for this machine.

If we assume that a memory reference takes 13 units of time, additions/subtractions take 8 units of time, and multiplies/divides take 32 units of time, then the number of units of time in our procedure is proportional to $24.5MN + 217.9M^2 + 199.6M + 99$. (A unit of time on a CDC Cyber 172 is 100 ns.) The number of iterations required to obtain a solution is dependent upon problem size (N), the amount of repulsion (R), and the stopping criterion value (ϵ).

TABLE I
NUMBER OF ARITHMETIC OPERATIONS PER STEP OF SOLUTION PROCEDURE

Step Number	Number Additions/Subtracts	Number Multiply/Divides	Number Memory Reference
1	$.1(4MN-\frac{M}{2}(M+7))$ $+.9(4M(M-1))$	$.1(3MN-\frac{3M}{2}(M+1))$ $+.9(\frac{3}{2}M(M-1))$	$.1(9MN-\frac{M}{2}(5M+13))$ $+.9(11/2M(M-1))$
2	$.9(2M(M-1))$	$.9(M(M-1))$	$.9(4(M(M-1))$
3	$2M$	0	$2M+2$
4	$2M$	0	$4M+2$
5	$4M$	$4M$	$10M+1$
6	$2M+1$	0	$2M+2$

D. Convergence

The question might arise, "does our method guarantee convergence of the force equations?" The answer is yes, because the Fx_i's and Fy_i's represent the partial derivatives of the potential energy function of the system of components, and since this system is dissipative in energy it implies that these derivatives are always pointing in the direction of lower potential energy. Our method always moves the solutions in the direction of these derivatives, hence leading to a convergent solution.

As an aside, we would like to mention that two other techniques for solving (8) were studied in depth. One used numerical integration, where a damping term was used to force a steady state solution. The second was an algebraic technique using the classical NR method. Neither of these techniques was as efficient as the one presented here. In Quinn [16] a solution technique using Fletcher–Reeves method of conjugate gradient minimization was presented. The method presented here produces a solution which is 2 to 3 orders of magnitude more efficient in terms of computer time. It has been called to our attention by one of the referees that the use of a procedure by Brown [17], which is a new variation on Newton's method, may lead to still more efficient computational results.

IV. Assignment of Components to "Real" Locations (Phase II)

The objective in solving the force equations was to find the correct relative position of every component with respect to every other component. In obtaining this solution we often find that the resultant placement is physically unacceptable from either the standpoint that the components overlap, or that the technology requires the components to reside in predetermined locations (slots). In an effort to resolve this apparent conflict we have broken the problem down into two subproblems, which are solved as follows: 1) if we are dealing with array boards, where components must go into slots, then an assignment technique is used; 2) if components are free to reside anywhere on the board, then a technique to resolve component overlap is used. For this latter case, the final placement is done manually, usually via an interactive graphic system.

A. Array Carrier

An array carrier is defined as a media having some number Q ($Q > M$) of predefined locations (slots) in which the M moveable components are to be placed. Generally, these slots are in a regular array (5×5, 5×10, 11×15, etc.), with distances between all columns being equidistant and distances between all rows being equidistant. This type of carrier is most often encountered when dealing with integrated circuit packages to be placed on a printed circuit board, in the design of master slice logic circuits, or in PLA's.

Since the results of Phase I give us the relative positioning of the components, we would like to distort this positioning as little as possible when assigning components to slots. That is, we would like to assign components to slots in such a way so as to minimize the total distortion of the placement. To accomplish this we have employed the linear assignment algorithm [8]. We have chosen to use Bourgeois and Lassalle's [11] implementation of Munkres' [8] formulation of the Hungarian Assignment algorithm in solving this problem. The assignment algorithm requires the generation of a cost matrix, where each element C_{ij} of the cost matrix represents the cost of assigning component j to slot i. We considered three cost functions, namely ones based upon rectilinear distance from the component to slot, the Euclidian distance and the square of the Euclidian distance.

Using rectilinear or Euclidian distance we can show analytically that the relative position of the components obtained in Phase I is not preserved. Therefore, we have chosen to use the square of the Euclidian distances. One reason for the success of the cost function can be seen from the mathematics involved. Note that even though we are physically moving components in two dimensions, when we use the assignment algorithm we are really working in M-dimensional space. That is, the linear assignment algorithm for assigning M components to Q slots ($M \leq Q$) dictates that we find a permutation $P = \{p_1, p_2, \cdots, p_M\}$ on the integers $1, 2, \cdots, Q$ taken M at a time such that $\sum_{i=1}^{M} C_{ip_i}$ is minimum over all such permutations. This sum represents the magnitude of an M-dimensional cost vector, and the distance of any M-dimensional vector in Euclidian space is equal to the square root of the sum of the squares of all its components. Therefore, since we are endeavoring to find the minimum of the total distance traveled by all components, we must compute the square root of the sum of the squares of all the distances traveled. Using this measure gives us the desired result for minimizing the total distortion of the components. It should be pointed out that others [2] who have attempted to use this type of technique have failed to realize the necessity of using this cost function. Hence, the quality of their results has been correspondingly reduced.

In Fig. 4 we show a placement obtained from Phase I, and in Fig. 5 the final placement obtained from Phase II. In Fig. 4(a) ((b)) we have drawn lines connecting those

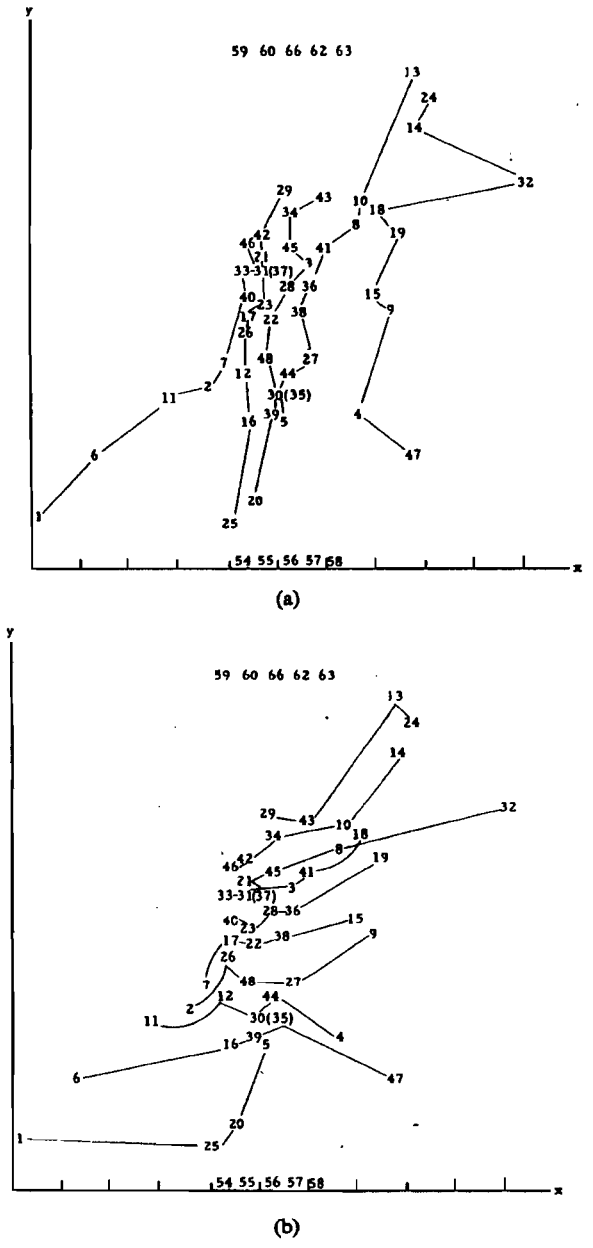


Fig. 4. (IC66) placement after Phase I. (a) Lines indicating columns from Phase II. (b) Lines indicating rows from Phase II.

components that are assigned to the same column (row) in Fig. 5. When rectangular or Euclidian distance measures are used, there are usually numerous intersections of lines implying that relative positions are not kept. This is not true when the square of the Euclidian distance is used.

Starting with the results shown in Fig. 4, the MST distances for the results of Phase II using rectilinear, Euclidian, and Euclidian distance squared are 963, 920, and 907 units, respectively.

Analysis and Comments on the Use of the Assignment Algorithm: One should note that Phase II of our process is conceptually quite different than Phase I. In Phase II, the interconnection topology of the components is no longer considered; only the physical positioning of the

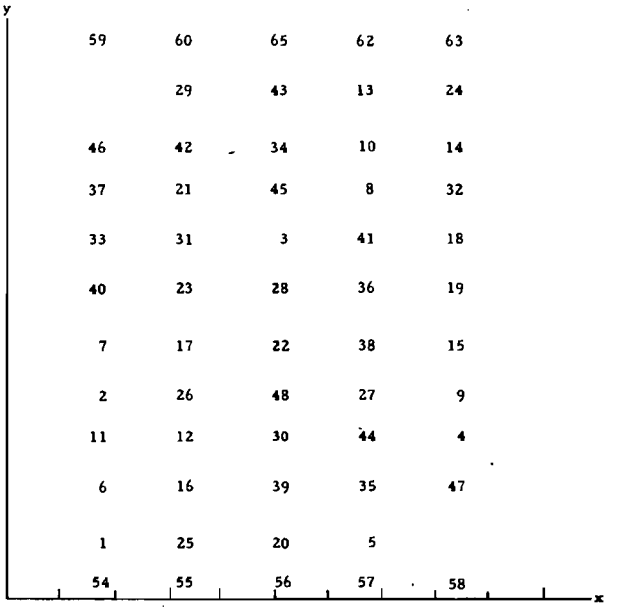


Fig. 5. (IC66) final placement after Phase II using Euclidian distance squared for assignment cost.

components is taken into account. This disregard for the interconnection topology can have some deleterious side effects. For example, if the results of Phase I dictate that the optimum board configuration is a 5×5 arrangement of slots, and our board consists of a 2×13 arrangement of slots, then the results of Phase II may not give the solution with the shortest total wire length. We have studied this situation for the case of a 5×5 lattice structure and have found that the difference in wire length between our 2×13 solution and the optimum 2×13 solution is less than 1 percent.

The computational complexity of the assignment algorithm is proportional to M^2Q. This implies that Phase II of the placement process requires more computation than Phase I, which is proportional to M^2.

B. Nonarray Carriers

Nonarray carriers are generally media containing a variety of different types and sizes of components. For example, an "analog" PCB usually contains IC's, transistors, capacitors, resistors, etc. In the past the layout of these types of carriers has traditionally been done manually. Components usually need not be aligned into columns and rows.

In the case where the assignment of components to slots is not a requirement, we must spread the components out over the board, eliminating any overlap that may result from Phase I. The elimination of overlap is done automatically by our Phase II program. It is carried out in such a way so as to preserve, as much as possible, the relative positioning of the components.

The implementation of our Phase I and II placement techniques has been embedded into an interactive graphics packaging program. This system allows the user to

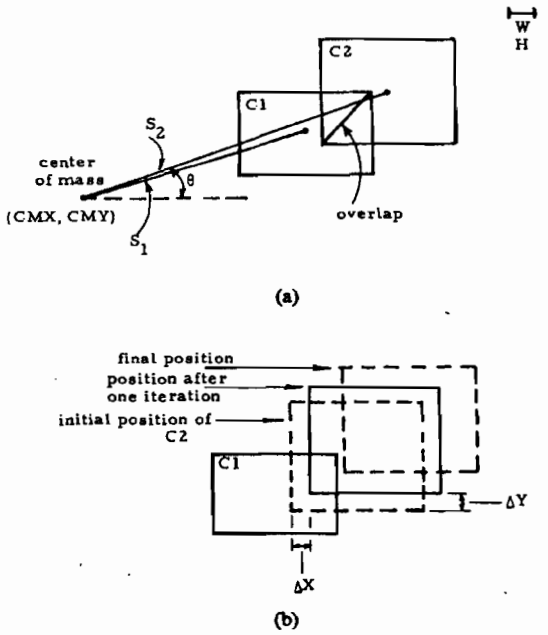


Fig. 6. Example of component overlap resolution.

view, via a vector graphics display, the positions of all components as found via our Phase I and II algorithms. It also allows the user to move any particular component to any position on the board by dragging it with a light pen. While moving the component, the program displays, in a "rubber band" fashion, the various places to which the component is connected. In this manner the user can easily find a good final position for all components. We have found this approach to be highly successful.

C. Resolution of Component Overlap

The resolution of component overlap is done on a pairwise basis. For M components, all $M(M-1)/2$ pairs of components are compared. If two components overlap, we move the component furthest away from the center of mass of the board, say component C, radially outward by some computed amount $(\Delta X_C, \Delta Y_C)$.

The set of pairs is processed iteratively. The process terminates when a sequence of $M(M-1)/2$ pairs of components are processed and no overlap exists.

In moving a component its overlap with another component may not be completely eliminated. In addition, new overlaps may be created when a component is moved. In any event, the process does readily converge and is effective in keeping the relative position of components in the final placement quite similar to the relative position as found by our Phase I algorithm.

In Fig. 6(a) we illustrate the initial location of two components $C1$ and $C2$, as well as the quantities S_1, S_2, θ and overlap. In Fig. 6(b) we show the result from one iteration of our procedure, as well as the final position obtained after several iterations.

Let W be one-half the width of the narrowest component, and H be one-half the height of the shortest component. Then the distance from the center of mass to the center of component Ci is $S_i = ((X_{Ci} - CMX)^2 + (Y_{Ci} - CMY)^2)^{1/2}$.

From Fig. 6(a), we see that $S_2 > S_1$, hence $C2$ is moved radially outward away from the center of mass. The quantities ΔX_{C2} and ΔY_{C2} are computed according to the equations

$$\Delta X_{C2} = \sigma_1 \cdot \min(\text{overlap}, W)$$

and

$$\Delta Y_{C2} = \sigma_2 \cdot \min(\text{overlap}, H)$$

where

$$\sigma_1 = \cos\theta = (X_{C2} - CMX)/S_2$$

and

$$\sigma_2 = \sin\theta = (Y_{C2} - CMY)/S_2.$$

The amount a component is moved in the x and y direction is restricted to be less than W and H, respectively, so that large components do not jump over smaller components. The sine and cosine functions are used so that components move out radially, i.e., in the direction of a vector from the center of mass to $C2$.

V. Experimental Results

In this section we will briefly review some of the results obtained by using this system. Throughout this chapter we will use the sum of the minimal spanning tree for all nets as a measure of the "quality" of a placement. For uniformity, the distance between each row and column in an array of slots will be taken as one unit.

A. Parameteric Analysis

The following list indicates several parameters of our procedure which we have studied:

1) initial placement of components,
2) problem size (number of components),
3) position of the center of mass (CM) of the moveable components,
4) repulsion constant,
5) interconnection matrix (how it represents the net list).

A complete analysis of our results can be found in Quinn [14]. In this section we will present a brief summary of the results obtained.

We used 12 test boards in evaluating our system. A few were "theoretical boards," constructed in such a way so that the optimal placement was known. Seven were real industrial boards. Boards labeled IC67, 108, 116, 136, and 151 are real boards taken from Stevens' [13] dissertation. Boards IC 67 and 151 were also processed by Hanan [12].

Initial Placement: It has been shown [12] that iterative improvement techniques tend to work better if preceded by some sort of constructive initial placement procedure (CIPP). This is also the case with our method. However, our CIPP is considerably different from the normally

recognized techniques. It consists of solving the force equations (8) using no repulsion ($R_{ij}=0\ \forall i,j$). The only forces keeping the system of components from totally collapsing are the center of mass forces. This CIPP is preceded by randomly placing the moveable components uniformly over the board on a continuous basis. By eliminating the R_{ij} term from (8), the resulting equations become linear and can be solved using classical techniques for the solution of simultaneous linear equations. Rather than develop this additional code, we chose to use our existing code to solve these equations while setting $R_{ij}=0$. This CIPP typically takes only a few (20–50) iterations to converge and takes considerably less computation per iteration than when repulsion is used. The CIPP phase takes about 2–8 percent of the total time required to find a placement.

Our experimental results indicate that the interconnection lengths obtained with and without using a constructive initial placement are approximately equal. This points out the relative insensitivity of this procedure to the initial positioning of the components. The most significant observation is that in almost all cases the procedure finds the same solution when our CIPP is used, independent of the initial random placement. The other very important observation, and in fact the reason for using CIPP, is that in most cases the total computer time required to obtain a solution is significantly reduced.

Problem Size: Problem size primarily affects the computational time for each iteration. Also the number of iterations for convergence appears to be more interconnection topology dependent than problem size dependent.

Position of the Center of Mass: The position of the *CM* in the vertical direction can have a pronounced effect upon the placement results. To see this, assume the connector is at the bottom of the board and recall that the position of the *CM* is at the geometrical center of the carrier to which components are to be assigned by Phase II. Note that all the components are held on the carrier away from the connectors. This positioning may distort the actual relative positioning in which the components would like to be arranged because the *FCMY* forces are excessively large. The large *FCMY* forces push those components that are weakly connected to the rest of the components further up on the carrier than required. The solution to this problem is to force the *CM* to a position that allows the components to end up as close to the connectors as possible without going below them. The determination of this position is possible by dynamically allowing the *CM* to move downward until the y position of the lowest moveable component goes to zero.

Phase I Stopping Criteria: The relative location process of Phase I is theoretically stopped when all the forces on the components are zero. This criteria is impractical from a computational standpoint. Therefore, we stop the process when

$$\sum_{j=1}^{N} \left[|Fx_i| + |Fy_i|\right] < \epsilon.$$

TABLE II
SOLUTION VERSUS-STOPPING CRITERIA

MST of \ ϵ	.001	.01	.1	1.0	10.0	100
IC136	2749	2762	2746	2738	2740	2763
IC151	2025	2025	2025	2034	2052	2053
IC67	713	713	713	719	764	775
2IC66	907	907	910	910	920	949
1L100	494	494	494	494	516	515
Number of Iterations						
IC136	456	366	279	193	49	14
IC151	388	319	239	162	93	35
IC67	184	151	120	90	52	5
2IC66	146	124	105	89	36	5
1L100	386	291	196	108	51	18

Our studies indicated that the number of iterations varied as a function of $(\log \epsilon)^{-1}$. Usually $\epsilon = 1.0$ produced the same results as those obtained using much smaller values of ϵ. Also for $\epsilon > 10$ the final results began diverging from the optimal. Some results are shown in Table II.

Repulsion Constant: The repulsive force between unconnected components is the single most critical parameter of the system. It affects the number of iterations as well as the quality of the results.

Evolution of the Equation for R: After much analysis and experimentation, it became evident that R had to be a function of M as well as the interconnection topology. For this reason R was made equal to $(1/C_R)\cdot(\Sigma_{i=1}^{N}\Sigma_{j=1}^{N}|K_{ij}|)/T$, (where T is the number of $K_{ij}=0$ terms), that is, R is proportional to the total attractive force and inversely proportional to the number of times R is used. This value for R has shown a great deal of insensitivity to different types of problems.

Determination of C_R: The method with which we chose to determine C_R was to make a series of runs on different problems, plotting C_R versus a number of normalized quantities such as 1) number of iterations for Phase I convergence, 2) MST of the result of Phase II with and without connector connections, and 3) run time of Phase I and Phase II combined. It is evident from our results that the optimum value of C_R is in the range $1 \leq C_R \leq 2$. It is still an open question as to exactly how to choose the optimum value of C_R for any given problem.

Interconnection Matrix (K_{ij}'s): The K_{ij}'s form the interconnection matrix and represent how the various components attract and repel each other throughout the relative location process. There are several ways of representing a signal set by the K_{ij}'s: 1) as a complete graph where every node is connected to every other node in the net, 2) as a MST, 3) as a string, where no node is connected to more than 2 other nodes, and 4) as a star, where all sink nodes are only connected to the source node.

All the figures and tables in this section were derived using the complete graph as representing the signal nets. For a net having S nodes, K_{ij} is set to $2/S$, because $2/S$ is that fraction of the $S(S-1)/2$ possible edges that are needed to interconnect the S nodes.

TABLE III.
COMPARISON OF OUR RESULTS WITH HANAN'S AND STEVENS'

QUINN-BREUER

Board Number	MST
IC67	710
IC108	1089
IC116	1519
IC136	2558
IC151	1915

STEVENS

Board Number	MST of CIPP	MST of PI	MST of SB
IC67	725	700	702
IC108	1230	1085	1122
IC116	1429	1320	1320
IC136	3306	2845	2733
IC151	2318	2243	2181

HANAN

Board #IC67	Initial Placement	Final Distance				
	Random	PI	NI	PIQ	NIQ	FDPRQ
10 Random Starts: Worst Dist.	1460	730	782	780	825	807
10 Random Starts: Avg. Dist.	1394	712	750	740	771	750
10 Random Starts: Best Dist.	1328	670	680	702	725	720
Constructive Dist.	812	691	682	717	726	695

Board #IC151	Initial Placement	Final Distance				
	Random	PI	NI	PIQ	NIQ	FDPRQ
10 Random Starts: Worst Dist.	4515	2122	2210	2240	2436	2255
10 Random Starts: Avg. Dist.	4427	2025	2120	2100	2252	2145
10 Random Starts: Best Dist.	4425	1890	1920	1997	2118	2026
Constructive Dist.	2046	2117	2170	2058	2126	2090

We investigated using the MST to represent signal nets. In about 60 percent of the cases there was some improvement, usually quite small, in using the MST representation. The CPU time increases significantly for this mode of operation, since the MST can only be computed after having first computed a solution using the complete graph. We therefore concluded that using the complete graph representation was the most desirable.

B. Comparison with Other Published Results

In [13], Stevens published net lists for five ILLIAC IV boards. He also gave the results for these boards using three different placement techniques, namely, a serial technique, a pairwise interchange (PI) technique, and the Steinberg (SB)–Rutman technique [9], [10]. Hanan [12] has also published results for two of these boards using a number of different methods, namely PI, neighborhood interchange (NI), force directed pairwise relaxation (FDPR), and SB. A description of each of these methods is given in [7]. For each of these methods, he makes a distinction between using the MST and the associated quadratic assignment problem (QAP). The acronym for each method is appended with a Q when QAP is used. Table III gives a comparison of our results with those of Stevens and Hanan. The results of Tables II and III differ because Table II was derived holding all parameter fixed except for ϵ, while Table III was derived using what was considered to be the optimum set of parameters for each board.

The results shown in Table III are self-explanatory. There is purposely no comparison given of computer time because all three sets of results were obtained using computers having considerably different characteristics.

C. Solution of Lattice Type Boards

We studied four boards having a lattice type interconnection structure and for which the optimal placement was known. In each case we obtained the optimal placement using our system. In a private communication with Dr. A. Patel of IBM he indicated that he processed two of these boards using both the classical pairwise interchange method and the SR procedure, and neither procedure produced an optimal result.

VI. Conclusions

We have presented a new formulation of the concept of force-directed placement, as well as an efficient procedure to solve the resulting mathematical equations. The technique gives excellent results both quantitatively, i.e., in finding the correct solution to known optimal problems, and qualitatively, when comparing with other published results. The procedure is economical to use, costing approximately $32 to place the IC151 board. This is equivalent to about 2 man-hours in cost.

At present, force directed procedures appear to be the most suitable types of procedures to offer the designer the capability of placing components on continuous boards. These procedures are also quite applicable for array type boards. Only in cases where the absolute minimum MST length is required should a method such as PI be used. From the results shown in Table III one can see that PI has the potential for giving better results than ours but at a great increase in cost. The PI results of Stevens [13] and Hanan [12] were obtained by taking the best result from ten random initial starting conditions. The average CPU times for PI on the IC151 board for Stevens' work on a Burroughs 6500 was 4020 s, and for Hanan's work on an IBM 360/91 was 485 s. Our procedure gives uniform results, independent of the initial starting, implying that one need make only one run rather than making multiple runs which is usually required if near optimal results are required when using PI. We estimate that our procedure is comparable in cost to the force directed pairwise relaxation technique of Hanan, and typically gives better results.

References

[1] V. Kevin and M. Whitney, "Minimum spanning tree," Algorithm 422, *Comm. ACM*, vol. 15, no. 4, pp. 273–274, Apr. 1977.

[2] F. T. Scanlon, "Automated placement of multi-terminal components," *Proc. Design Automation Workshop*, pp. 143–154, July 1971.

[3] N. R. Crocker *et al.*, "Computer aided placement and routing of high density chip interconnection systems," International Computers Limited, Microsystems Division, Research and Advanced Development Organization Tech. Report, Wenlock Way, West, Gordon, Manchester M12 5DR, England, Aug. 1972.

[4] C. J. Fisk, D. L. Caskey, and L. E. West, "ACCEL: Automated circuit card etching layout," *Proc. IEEE*, vol. 55, pp. 1971–1982, Nov. 1967.

[5] M. Hanan and J. M. Kurtzberg, "Placement techniques," in *Design Automation of Digital Systems: Theory and Techniques*, Ed. M. A. Breuer. New York: Prentice-Hall, 1972, ch. 5.

[6] P. A. Stark, *Introduction to Numerical Methods*. New York: Macmillan, 1970.

[7] *Design Automation of Digital Systems: Theory and Techniques*, Ed. M. A. Breuer. New York: Prentice-Hall, 1972.
[8] J. Munkres, "Algorithms for the assignment and transportation problems," *J. SIAM*, vol. 5, pp. 32–38, 1957.
[9] L. Steinberg, "The backboard wiring problem: A placement algorithm," *SIAM Review*, vol. 3, no. 1, pp. 37–50, Jan. 1961.
[10] R. A. Rutman, "An algorithm for placement of interconnected elements based on minimum wire length," *Proc. 1964 SJCC*, pp. 477–491.
[11] F. Bourgeois and J. C. Lassalle, "An extension of the Munkres algorithm for the assignment problem to rectangular matrices," *Comm. ACM*, vol. 12, pp. 802–806, Dec. 1971.
[12] M. Hanan, P. R. Wolff, and B. J. Agule, "A study of placement techniques for computer logic graphs," *Proc. 13th Design Automation Conf.*, June 1976, pp. 214–224.
[13] J. E. Stevens, "Fast heuristic techniques for placing and wiring printed circuit boards," Ph.D. thesis, Computer Science Dept., University of Illinois, Urbana, 1972.
[14] N. R. Quinn, Jr., "An analysis of a forced directed component placement procedure for printed circuit boards," Ph.D. dissertation, University of Southern California, Jan. 1977.
[15] K. M. Hall, "An r-dimensional quadratic placement algorithm," *Management Sci.*, vol. 17, pp. 219–229, Nov. 1970.
[16] N. R. Quinn, Jr., "The placement problem as viewed from the physics of classical mechanics," *Proc. 12th Design Automation Conf.*, pp. 173–178, June 23–25, 1975.
[17] K. M. Brown, "A quadratically convergent Newton-like method based upon Gaussian elimination," *SIAM J. Numerical Analysis*, vol. 6, pp. 560–569, Dec. 1969.

Neil R. Quinn, Jr., was born in Los Angeles, CA, on February 15, 1945. He received the B.S. degree in physics from Loyola University of Los Angeles, CA, in 1967, and the M.S. degree in computer science and the Ph.D. in electrical engineering from the University of Southern California, Los Angeles, in 1970 and 1977, respectively.

He worked at General Dynamics Corporation in Pomona, CA, from 1967 to 1979. He was involved in the areas of computer aided design and manufacturing and hybrid computing until 1975. In 1976, he was appointed Supervisor of Hybrid Computing Development. In 1976, he was appointed Chief of Engineering Data Systems which involves directing all of the Engineering computing activities for the Pomona Division of General Dynamics. He is presently Vice-President of systems and operations at Environmental Management Systems, Inc., and is working on problems in the area of computer aided energy management. In addition, he is involved in consulting activities in the areas of electronic packaging.

Dr. Quinn is a member of the Association for Computing Machinery.

Melvin A. Breuer (S'58–M'65–SM'73) was born in Los Angeles, CA, on February 1, 1938. He received the B.S. degree in engineering with honors from the University of California, Los Angeles, in 1959, and the M.S. degree in engineering, also from the University of California, Los Angeles, in 1961. In 1965 he received his Ph.D. degree in electrical engineering from the University of California, Berkeley.

In 1965 he joined the staff of the Electrical Engineering Department of the University of Southern California, Los Angeles, where he is currently a Professor. His main interests are in the area of switching theory, computer aided design of computers, fault test generation, and simulation. He is the editor and coauthor of *Design Automation of Digital Systems: Theory and Techniques* (Prentice-Hall); editor of *Digital System Design Automation: Languages, Simulation and Data Base*, Computer Science Press; coauthor of *Diagnosis and Reliable Design of Digital Systems*, (Computer Science Press); and editor-in-chief of the *Journal of Design Automation and Fault Tolerant Computing*. In addition to his research activities, he has been at the forefront of developing courses on design automation and fault tolerant computing at universities and at a number of research institutes.

Dr. Breuer is a member of Sigma Xi, Tau Beta Pi, and Eta Kappa Nu.

TAB 40

HALO:AN EFFICIENT GLOBAL PLACEMENT STRATEGY FOR STANDARD CELLS

Yeong-Yil Yang and Chong-Min Kyung
Department of Electrical Engineering
Korea Advanced Institute of Science and Technology
P.O. Box 150, Cheongryang, Seoul, Korea

Abstract

This paper describes an efficient global cell (module) placement strategy called HALO (Hierarchical Alternating Linear Ordering) which generates a global 2-D placement of circuit modules by hierarchical application of linear ordering in alternating direction. It is explained why HALO *should* perform better than other typical, somewhat successful, analytical approaches such as min-cut, force-directed relaxation (FDR) or its likes. We have applied the HALO algorithm for standard cell placement. Experimental results on two benchmark circuits, *primary1* and *primary2* consisting of 752 and 2907 cells have shown a decrease of the half-perimeter routing lengths by 7% and 24% respectively, compared to the best available results[6] obtained so far. Total CPU time including the following detailed placement was less than half of the earlier work[6].

1. Introduction

The problem of finding the optimal placement of circuit modules in 2-D space with various objectives such as chip area, wire length, etc. has been tackled by many approaches. As the number of circuit modules (cells) in a circuit is drastically increased, IC designers are now in greater demand of a fast, high-performance placement packages than ever before. Simulated annealing has shown a better performance than other heuristic approaches for the standard cell placement, but only with enormous amount of CPU time.[1] Other heuristic algorithms such as min-cut[2,3], FDR (Force-Directed Relaxation)[4], and its variations[5,6,7,8] have been quite successful in itself, and have helped us in understanding the interal mechanism of the 2-dimensional circuit placement process. In this paper, we will briefly mention the major drawbacks of these heuristic algorithms and explain how these problems are handled in our algorithm called HALO (Hierarchical Alternating Linear Ordering).

Min-cut is basically a method for successively confining the modules in each compartment until each includes only one module. The only criteria of dividing the whole modules into two groups is just number of cuts, ignoring the global connectivity among modules. Therefore, it is a pure top-down approach where the decision in the upper-level is made with very little information or anticipation on the lower level decision. On the other hand, FDR tries to *order* the modules directly in 2-D space by using spring-force relaxation analogy. The resultant placement, therefore, *should* be more global than the result of min-cut. However, most experimental results have shown the superiority of min-cut. The reason is due to the fact that FDR completely ignores the size of the modules, while min-cut considers the size information. There are some variational forms of FDR reported[6,8] which have tried to take the module size into consideration while still preserving the FDR's inherent nature of 'globality'. For example, Tsay *et. al*[8] proposed performing the FDR processes in each of the partition of the whole chip with some coordination between them by using the concept of 'floating pin' which is similar to 'terminal propagation'[3] in min-cut. A closer coordination between two FDR processes in each chip partition using constrained optimization technique was reported by Kleinhans *et. al.*[6] The experimental results of Kleinhans *et. al*[6] on the benchmark circuits are better than any other published results performed on the same circuits in terms of the half-perimeter wire length estimation.

In this paper, we propose an algorithm called HALO (Hierarchical Alternating Linear Ordering) for the global placement of standard cells. (It is assumed that the whole placement process is divided into two steps: global placement where relative module positions are obtained where some inter-module overlaps are allowed, and detailed placements, or packing where such overlaps are removed.) In section 2, we describe the algorithm HALO and explain why HALO should outperform other heuristic algorithms such as min-cut and FDR and their variations, especially the GORDIAN.[6] We have implemented HALO in our program for standard cell placement. The remaining procedures in the standard cell placement, i.e., row assignment, feedthrough cell assignment (We chose to perform feedthrough assignment during the placement, rather than during the global routing process in order to minimize the effect of the feedthrough cells on the eventual width of each row.) and the intra-row cell assignment will be explained in section 3. Finally experimental results are given with discussions in section 4.

2. HALO (Hierarchical Alternating Linear Ordering) for global placement

The overall procedure for the standard cell placement using HALO as a global placement strategy is shown in Fig. 1. HALO receives the circuit connectivity as its input data and calculates the center-of-mass positions of the circuit modules. After some number (typically 2 or 3) of iterations of row assignment and feedthrough cell assignment, the whole process is finished by performing the intra-row cell placement. Fig. 2(a), (b), (c) and (d) show, in succession, how the cells are confined to each of the two (not necessarily two, but it can be three, depending on the aspect ratio of the partition in question) subregions using linear ordering. The criteria used in the linear ordering of cells will be explained later in this section.

Let us assume that the cells are linearly ordered as shown in Fig. 2(a). We divide the chip area into two subregions such that each can accommodate each (left and right, in this case of horizontal partition) group of cells. Usually the two groups are divided such that the sum of areas of the modules in each group are approximately the same. In Fig. 2(a), the modules represented as circles are permanently assigned to L-partition, so do those represented as boxes to R-partition. It is very important to note that in dividing the modules into two partitions as in Fig. 2(a), all the circuit connectivity information has been reflected. This needs to be contrasted to min-cut which only minimizes the number of cuts. We consider min-cut as pure top-down strategy where bottom-level facts are ignored in the top level decision, being compared to our strategy where the bottom-level information is reflected in top-level decision to the extent

CH2868-8/90/0000-0448$1.00 © 1990 IEEE

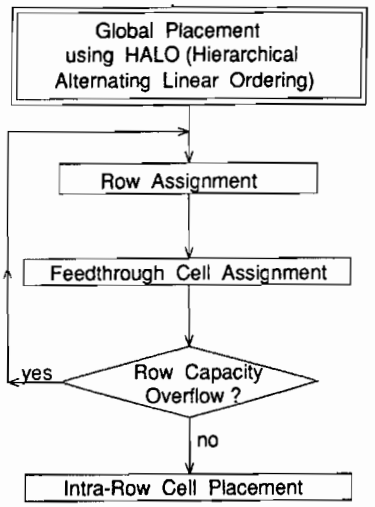


Fig. 1 Overall standard cell placement flowchart using HALO for global placement

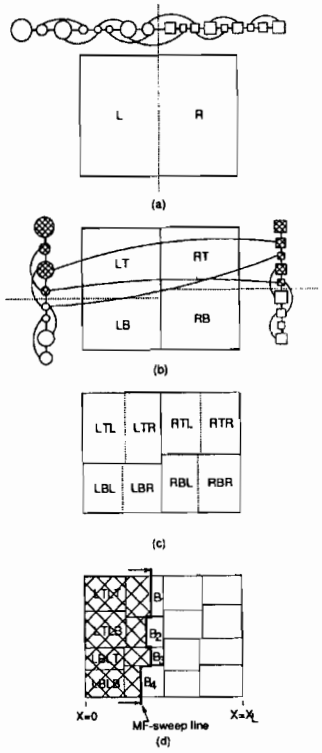


Fig. 2 HALO procedure (a) H1-ordering (b) V1-ordering (c) H2-ordering (d) V2-ordering is finished. H3 ordering is in progress using multi-frontal (MF) sweep line.

the bottom-level information can be known at the time of top-level decision. Compared with the FDR-based approaches,[6,8] HALO considers the size of the modules and the inter-module connectivity simultaneously. In GORDIAN[6], the module sizes are introduced too late, i.e., only after the dot (of size zero) distribution using constrained optimization ignoring the module size is obtained. Moreover, the constrained relaxation of module positions in 2-D is redundant, because only the ordering in one direction is utilized in performing the partition in any level of the hierarchy of the procedure.

Once the modules are assigned to L- and R-partition using the horizontal ordering which we call H1-ordering which means the 1st ordering in horizontal direction, V1-ordering is performed as shown in Fig. 2(b), in both the L- and R-partition, simultaneously. The connectivity among modules in L- and R-partitions still exist, although the modules belong to different partitions. After V1-ordering, the shaded circles and shaded boxes are confined to LT- and RT-partition, respectively, while empty circles and empty boxes are confined to LB- and RB-partition, respectively. Fig 2(c) shows the result of H2-ordering in a similar way. Let us assume now that all the cells are V2-ordered, and we want to perform H3-ordering. Fig 2(d) shows a snapshot during the H3-ordering process. Assume that the cells in the shaded region are already H3-ordered and those in the right side of the multi-frontal (MF) sweep line are V2-ordered. (H3-ordered cells denote those that are already ordered in the H1, V1, H2, V2 and H3-ordering, while V2-ordered cells denote those ordered in the H1, V1, H2 and V2-ordering.)

Next, we will briefly describe the linear ordering process which is bassed on the method proposed in [9]. As shown in Fig. 3, the cells are classified into three groups during the ordering process, i.e., Ordered(O), Active(A) and Unordered(U) groups. Initially, all the cells are in U-group and they are moved, one by one, into the O-group. The cells which are not so far ordered but have nets in common with the cells in O-group are moved into the A-group. Only the cells in A-group become the candidates to be selected as the next cell in the linear ordering process. Nets also can be classified into three groups during the ordering process: i) *New net* is connected to none of the cells in O-group. ii) *Terminating net* is the net having connection between only one cell in A-group and one or more cells in O-group. iii) *Continuing net* is the net having connection among one or more cells in O-group and two or more cells in A-group.

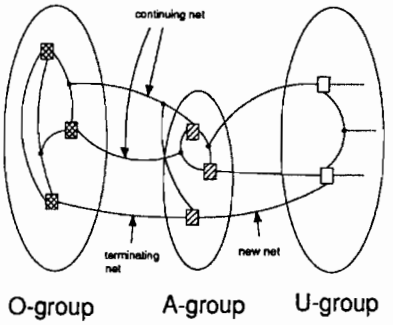


Fig. 3 The status of the cells during the ordering

Algorithm 1 is a heuristic algorithm for finding the linear order of cells. The inputs to this procedure are a set of regions whose area is equal to the summation of the area of cells assigned to those regions, the cells contained in each of the regions, and the ordering direction. During the ordering, the regions are classified into three groups. All the cells in Ordered Region(O-region) such as LTLT, LTLB, LBLT and LBLB in Fig. 2(d) are ordered. If none of the cells in a region are ordered, the region is called Unordered Region(U-region). The region where only part of the cells are ordered is called Active Region(A-region). We define the i-th point B_i as the x-coordinate value of the vertical interface line which is the boundary between the ordered cells and unordered cells in the i-th region as shown in Fig. 2(d).

Initially, all the B_i's are set to the leftmost x-value (lowermost y-value) of the regions, if the ordering direction is horizontal (vertical). A cell in the region having the minimum B_i is selected according to the linear ordering procedure explained in Fig. 3 and the following selection criteria, and B_i is updated accordingly. Followings are the cell selection criteria in the order of the priority.

1) Select a cell c_i such that the number of nets connecting the O-group and the other groups is minimized after the movement of the cell c_i. (I.e., select a cell c_i such that the difference between *terminating nets* and *new nets* of the cell c_i is maximum.)
2) Select a cell c_i such that the number of terminating nets connected to c_i is maximum.
3) Select a cell c_i such that the number of continuing nets connected to c_i is maximum.

Algorithm: **Finding_Cell_Order**
Inputs: a circuit, a set of regions, the direction of ordering (horizontal or vertical)
Outputs: Linear order of cells

Step 1) Set O-group, A-group, O-region, and A-region empty. Insert all the cells into U-group and calculate B_i's for the regions. Insert those regions, r_i's into A-region, which have the minimum value of B_i.
Step 2) Select the regions r_i's in A-region whose length is minimum (for example, B_4 in Fig. 2(d)) and then select a cell c_i among the cells in the region r_i. If the region r_i has no cells in A-group, then select the cell c_i having the least connections with other cells, else select a cell c_i in A-group according to the cell selection criteria mentioned above. Move a cell c_i into O-group and move all the cells connected to c_i from U-group to A-group. The length of the region r_i, B_{r_i} is increased by the amount of the area occupied by the cell c_i divided by the height (width) of the r_i, when the ordering direction is horizontal (vertical). If there are still unordered cells in the region r_i, repeat step 2.
Step 3) Remove the region r_i from A-region and insert all the regions adjacent to the region r_i to A-region. If A-region is empty, then stop, else go to step 2.

Algorithm 1: An algorithm for finding the linear order of cells

3. Detailed Placement for Standard Cells

After the global placement using the HALO procedure is performed, the center-of-mass positions of each cell are obtained. To obtain the final placement of standard cells, the cells should be, first, assigned to their rows and, secondly, positioned within the row. Besides, feedthrough cells should be added if necessary. In this section, the procedure for obtaining the final standard cell layout from the result of HALO procedure is explained.

Feedthrough cells in the standard cell layout are added to complete the connections for a net. Fig. 4 shows the center-of-mass positions of cells with the row assignment where the cells represented as black circles belong to a common net, n_i. Assume that the i-th (k-th) row is the nearest one from the j-th row among all the rows above (below) the j-th row having at least one cell connected to a net n_i. A feedthrough cell f_i is then introduced in the j-th row for the complete connection of the net n_i, and positioned in the j-th row according to the distribution pattern of the cells in i-th row and k-th row. Algorithm 2 is a heuristic called **Row_Assignment** for assigning the cells and feedthrough cells to the rows. In this procedure, the iterations from step 1 to step 3 are necessary to make the widths of the rows more uniform, because it is hard to predict the number of feedthrough cells required after the row assignment. In each iteration, the width of each row is calculated as the summation of the widths of the cells assigned to that row in the current iteration and the widths of feedthrough cells assigned to that row in the previous iteration.

Algorithm: **Row_Assignment**
Inputs: the center-of-mass positions of cells, result from HALO procedure

Step 1) Sort the cells in the non-increasing order of y-coordinate values and name it CELL_LIST. For each row r, set n_r (the number of feedthrough cells assigned to the row r) and f_r (the flag to indicate whether the row r is filled or unfilled) to zero and unfilled, respectively.
Step 2) For the cell c_i at the front of CELL_LIST, find the row r such that r is the nearest row from the position (y-coordinate value) of the cell c_i and f_r is unfilled. If the summation of the widths of the cells assigned to the row r and the width of a feedthrough cell multiplied by n_r is enough to fill up the row r, set f_r as filled. Delete the cell c_i from CELL_LIST. If there are remaining cells in CELL_LIST, go to the beginning of step 2.
Step 3) For each row r, set the number of the feedthrough cells required in the row r to n_r. Calculate the width of the row which is the summation of the widths of the cells and the feedthrough cells. If the difference of the widths between the shortest row and longest row is greater than some specified value, go to step 1.
Step 4) For each feedthrough cell, determine the position of the feedthrough cell. Stop.

Algorithm 2: An algorithm for the row assignment and feedthrough cell assignment

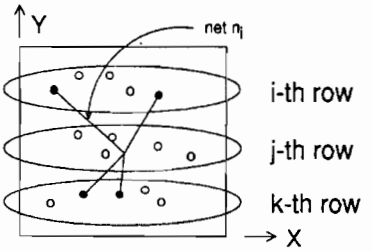


Fig. 4 The center-of-mass positions of cells (the result from HALO procedure) with row assignment where the cells represented as black circles belong to a common net, n_i.

After the cells are assigned their rows using the above-mentioned row assignment, the positions of the cells and feedthrough cells within the row need to be determined. A heuristic algorithm which is a modification of the heuristic, **Finding_Cell_Order** in the previous section is used for this purpose. The ordering scheme for intra-row assignment is quite similar to the ordering scheme for the row assignment explained earlier. However, there are several differences to note. One is that only the specified number, K, of cells reside in the active cell list, while all the cells connected to the *placed* cells reside in the active cell list in the ordering described in the previous section. The other is that when a cell c_i in the row r is placed, the cell having the least x-value among *unplaced* cells in the row r is moved to the active cell list of the row r.

The computational complexity of the whole standrad cell placement procedure is $O(n(\log n)^2)$ and Fig. 5 shows the exper-

imental results of computation time of the proposed algorithm as a function of the number of cells. It has shown approximately a $O(n \log n)$-time behavior.

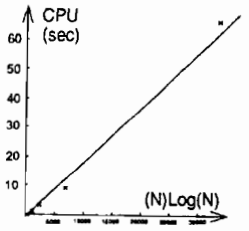


Fig. 5 The computation time versus the circuit size (the number of the cells), where the number of the cells are scaled by $n \log n$. The numbers of the cells of the circuits used in the experiments are 67, 144, 270, 752, and 2907.

4. Experimental Results and Discussions

The proposed algorithm, HALO (Hierarchical Alternating Linear Ordering) was implemented as a global placement package for standard cells using C language in SUN4/UNIX. Table 1 shows the characteristics of the benchmark circuits, *primary1* and *primary2*.[10] Table 2 shows the comparison of the half-perimeter routing length of the benchmark circuits. The cells of the circuits, *primary1* and *primary2*, were placed in 17 and 26 rows, with estimated channel widths of 220*um* and 270*um*, respectively. The data of the first three methods, i)min-cut with terminal propagation,[3] ii) the relative/transportation method[11], and iii) GORDIAN[6] are obtained from [6]. The half-perimeter routing length of the proposed algorithm is calculated with, and without, inserting necessary feedthrough cells. The value in our result outside (inside) the parenthesis is the half-perimeter routing length with (without) feedthrough cells. The positions of I/O cells are assumed to exist at the pad positions being given in the benchmark data. The resulting length include the wires to these I/O cells. As shown in Table 2, the half-perimeter routing length of the proposed algorithm for the circuits,[10] *primary1* and *primary2*, is less than that of GORDIAN[6] about 7% and 24%, respectively. Our result has shown a 4% and 17% of improvement compared to [6], even after including the feedthrough cells. Table 3 shows the placement results for the circuits in Table 1. Fig. 6 shows a sample layout of the placement of *primary1* in 16 rows using a global and channel router. These router's performance were not optimized in terms of the resultant number of tracks. For the assessment of the quality of a circuit placement, we believe that the half-perimeter cost is a reasonably good estimate. In the placement of standard cells of the circuits, *primary1* and *primary2*, feedthrough pins within a cell were not used.

In conclusion, we proposed a fast heuristic algorithm, HALO for the global placement of standard cells, which should actually outperform in principle and have outperformed previous heuristic algorithms in many experiments. The CPU time for the *primary1* and *primary2*, run on SUN4 (7MIPS machine) were 9.43 sec and 65.50 sec, respectively, which shows that the proposed HALO approach has merits in the sense of CPU time over the previous approaches.

circuit	Number of cells	Number of nets
primary1	752	904
primary2	2907	3029

Table 1. Statistical data for benchmark circuits.

algorithm	half-perimeter routing length [m]	
	primary 1	primary 2
Min-Cut	1.739	9.823
RT	2.177	8.685
GORDIAN	1.503	8.142
ours	1.439(1.397)	6.73(6.169)

Table 2. Half-perimeter routing length comparison for the circuits in Table 1, where the value in the parenthesis is the half-perimeter routing length when feedthrough cells are not inserted.

Circuit	Number of rows	Number of feedthroughs	Total number of channels
primary1	16	576	300
primary1	17	613	319
primary2	26	3720	993

Table 3. Placement results for the circuits in Table 1.

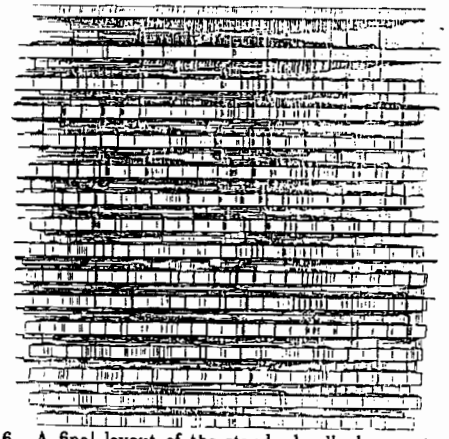


Fig. 6 A final layout of the standard cell placement of the circuit called *primary1* (first circuit in Table 1)

References

[1] C. Sechen and A. Sangiovanni-Vincentelli, "TimberWolf 3.2:A new standard cell placement and global routing package," *Proc. 23rd DAC*, pp. 423–439, June 1986.

[2] M. Breuer, "Min Cut Placement," *J. Design and Fault Tolerant Computing*, 1, (4), pp. 343–363, Oct. 1977.

[3] A. E. Dunlop and B. W. Kernighan, "A Procedure for Placement of Standard-Cell VLSI Circuits," *IEEE Trans. on CAD of IC's and Systems*, Vol. 4, No. 1, pp. 92–98, Jan. 1985.

[4] N.R. Quinn and M.A. Breuer, "A Force-Directed Component Placement Procedure for PCB's," *IEEE Trans. on CAS*, Vol. 26, No. 16, pp. 663–670, June 1979.

[5] G. J. Wippler, M. Wiesel, and D. A. Mlynsky, "A Combined Force And Cut Placement For Hierarchical VLSI Layout," *Proc. 19th DAC*, pp. 617–677, 1982.

[6] J. M. Kleinhans, G. Sigl, and F. M. Johannes, "GORDIAN:A New Global Optimization/Rectangle Dissection Method for Cell Placement," *ICCAD-88*, pp. 506–509, 1988.

[7] C. K. Cheng and E. S. Kuh, "Module placement based on resistive network optimization," *IEEE Trans. on CAD of IC's and Systems*, Vol. 3, No.3, pp. 218–225, July 1984.

[8] R. S. Tsay, E. S. Kuh and C. P. Hsu, "Module placement for large chips based on sparse linear equation," *Int'l J. of Circuit Theory and Applications*, Vol. 16, pp. 416–423, 1988.

[9] Sungho Kang, "Linear Ordering and Application to Placement," *Proc. 20th DAC*, pp. 457–463, 1983.

[10] B. Preas and K. Roberts, *Physical Design Workshop*, Hilton Head, South Carolina, 1987.

[11] K. M. Just, J. M. Kleinhans, and F. M. Johannes, "Relative Placement and the Transportation Problem for Standard-Cell Layout," *Proc. 23rd DAC*, pp. 308–313, 1986.

TAB 41

An Efficient Algorithm for the Two-Dimensional Placement Problem in Electrical Circuit Layout

SATOSHI GOTO, MEMBER, IEEE

Abstract—**This paper deals with the optimum placement of modules on a two-dimensional board, which minimizes the total routing length of signal sets. A new heuristic procedure, based on iterative improvement, is proposed. The procedure repeats random generation of an initial solution and its improvement by a sequence of local transformations. The best among the local optimum solutions is taken as a final solution. The iterative improvement method proposed here is different from the previous one, in the sense that it considers interchanging more than two modules at the same time and examines only a small portion of feasible solutions which has high probability of being better. Experimental results show this procedure gives better solutions than the best one up to now. The computation time for each local optimum solution grows almost linearly with regard to the number of modules.**

I. Introduction

IN THE design of a large scale electronic system, one of the most important problems is the layout problem, which involves the placement and interconnection of a large number of components and subsystems to satisfy a given specification. The problem is of particular significance in the present day design of VLSI chips, as a huge number of components is mounted on one chip and interconnected more complicatedly. Because of its complexity, the layout problem is usually divided into two stages, namely: "placement" and "routing." In the placement stage, the locations of individual modules are decided on a board to facilitate routing. In the routing stage, the interconnections on external leads, called pins, are made in such a manner as to satisfy various physical constraints.

This paper will deal with only the placement problem. The final layout design goal is to achieve 100-percent routing. However, it is far too difficult to consider the final goal itself in the placement stage. The usual goal will be adopted of minimizing the total routing length defined in an appropriate way. From the computational complexity point of view, this problem is considered to be a hard combinatorial problem in the sense that the computation time required to obtain the real optimum solution increases in exponential order when the problem size, i.e., number of modules, increases. Hence, algorithms based on heuristic rationales have to be employed.

Manuscript received November 30, 1979; revised May 29, 1980.
The author is with Central Research Laboratories, Nippon Electric Company, Ltd., Kawasaki, 213 Japan.

Until now, several algorithms have been proposed. All of these algorithms are either constructive or iterative [1]–[7]. Hanan and Kurtzberg survey known algorithms in [3] and showed some experimental results in [7]. They concluded that the force-directed pairwise relaxation algorithm, called FDPR, operating on the associated quadratic assignment problem, yielded the best result in near minimum time.

This paper proposes an algorithm which is more efficient than the best one up to now. The algorithm, as usual, consists of two phases, the initial placement phase, called SORG, and the iterative improvement phase, called GFDR. The Sub-Optimum-Random-Generation (*SORG*) method sequentially selects unplaced modules according to their connectivity to other modules and places them so as to minimize the total routing length. The SORG method differs from those in pertinent literature [3], in that it produces a good initial solution randomly by local random selections. The Generalized-Force-Directed-Relaxation (*GFDR*) method performs iterative improvement to reduce the total routing length. It is different from the FDPR method in [7], which tries to interchange only a pair of modules, whereas the GFDR method interchanges more than two modules at the same time. The GFDR method is considered to be a generalization of the FDPR, combined with the technique of realizing a λ-optimum solution proposed in [8].

II. Preliminaries

Consider a two-dimensional board on which modules are to be placed. The *board* is characterized in terms of a finite array of slots. A matrix location, or *slot* may be represented by a point in an x–y coordinate system. *Modules* are the entities which are to be assigned to slots on the board. It is required that one module can occupy one and only one slot, and on each slot not more than one module is allowed.

Modules contain pins for connection by physical wires to form *signal sets*. In this study, the pins on the modules are ignored and the distance is measured from the center of the module. Hence, a signal set becomes a subset of modules, and a signal set specification defines the connection of all modules on a specific board.

Let the board have m rows and n columns. It may be assumed that the number of modules is equal to $m \times n$

0098-4094/81/0100-0012$00.75

without loss of generality, because dummy modules, which are not connected, can always be introduced. The distance between two adjacent slots is defined as, either vertical or horizontal, one unit length.

The routing length of a signal set is defined as half-perimeter of the smallest rectangle, which encloses the modules in the signal set [5], [6]. The placement problem is now defined in the following.

> Given a set of modules with signal sets defined on subsets of these modules and a set of slots, place all the modules on the slots so that the total routing length over all signal sets is minimum.

III. Median of a Module

Preceding a placement algorithm, it is necessary to introduce a concept, called median of a module, and present an algorithm to find it, since the present placement algorithm depends on it. Let us consider a board on which every module is placed. Pick one module, denote it by M. Move only module M on the board, while the other modules remain fixed. The routing length of a signal set does not change, as long as the signal set is not connected to module M. Therefore, consider the signal sets connected to module M only and the sum of the routing length of these signal sets. This value is referred to as *the routing length associated with module M*.

Now, define the median of module M. Module M may be placed on $m \times n$ different positions. *The module M median* is defined as a position where the routing length associated with module M is minimum. Next, sort all the routing lengths associated with module M with respect to the module M position in ascending order. In this order, choose ϵ elements from the minimum one. The set of these ϵ positions is defined as the *ϵ-neighborhood for module M median*.

Now consider how to find a median of a module. Let i $(i=1,2,\cdots,r)$ designate a signal set which is connected to module M. For each signal set i, consider the smallest rectangle which encloses the module in the signal set. Here, module M is excluded from the signal set when forming the rectangle. Let us denote the rectangle by l_i and its figure by parameters (x_i^a, y_i^a) and (x_i^b, y_i^b), where x_i^a and x_i^b are the minimum and maximum values in the x-direction on the rectangle, respectively. The same definitions are pertinent for y_i^a and y_i^b in the y-direction.

The routing length associated with module M, which is required to place module M in position (x, y), is given by

$$F(x,y)=\sum_{i=1}^{r}\left(f_i(x)+f_i(y)\right) \tag{1}$$

where

$$f_i(x)=\begin{cases} x_i^a-x, & x<x_i^a \\ 0, & x_i^a \leqslant x \leqslant x_i^b \\ x-x_i^b, & x>x_i^b \end{cases} \tag{2}$$

$$f_i(y)=\begin{cases} y_i^a-y, & y<y_i^a \\ 0, & y_i^a \leqslant y \leqslant y_i^b \\ y-y_i^b, & y>y_i^b. \end{cases} \tag{3}$$

The problem is to find a pertinent position (x, y) on the board, such that $F(x, y)$ is minimized. Since the function $F(x, y)$ has a separable form with respect to variables x and y, $F(x, y)$ can be calculated independently from each other for x and y. The y-component can be found in the same way as the x-component. Thus only the x-component will be discussed in the following.

Equation (2) is transformed as

$$f_i(x)=\frac{1}{2}\left\{|x-x_i^a|+|x-x_i^b|-\left(x_i^b-x_i^a\right)\right\}. \tag{4}$$

The problem is reduced to finding a position where

$$\sum_{i=1}^{r}\left(|x-x_i^a|+|x-x_i^b|\right)$$

is minimum, since $x_i^b-x_i^a$ is a constant value. The value $|x-x_i^a|+|x-x_i^b|$ indicates the sum of the distances from x to x_i^a and x to x_i^b, thus the problem is to find a point x such that the total sum of the distances from x to each point x_i^a, x_i^b $(i=1,2,\cdots,r)$ is minimum.

In general, it is necessary to solve the following problem. There are α_i points on position i $(i=1,2,\cdots,n)$ along the line. Find a position x on the line such that

$$\sum_{i=1}^{n}\alpha_i|x-i|$$

is minimum. This problem is a particular case of finding an absolute median of a graph presented in [9]. The present problem, treating only a linear tree instead of a general graph, can be easily solved by using the following theorem.

> *Theorem 1:* Point q is the median if
>
> $$\sum_{i=1}^{q-1}\alpha_i<\frac{N}{2}\leqslant\sum_{i=1}^{q}\alpha_i \tag{5}$$
>
> holds, where $N=\sum_{i=1}^{n}\alpha_i$.

Fact 1: The total distance decreases monotonically from point 1 to a median. From a median to point n, it also increases monotonically. This fact is quite useful when more than one median are interested.

Fact 2: The x-component and y-component of a median can be calculated independently from each other. Each component has the characteristics mentioned in Fact 1. Therefore, the median on the two-dimensional board can be found easily. When interest is in finding the kth minimum, instead of the first minimum, it is possible to use the efficient algorithm reported in [10]. Thus the ϵ-neighborhood of a module can be easily obtained.

IV. A Placement Algorithm

The problem treated here is considered to be NP-complete[1] in the sense of Cook and Karp [11]. Hence, some heuristic procedure should be applied in order to obtain a nearly optimum solution in reasonable computation time.

There are, in general, two types of heuristic methods for this kind of problem. One is a *constructive method*, which obtains a solution using heuristic rules, often in sequential, deterministic manner. The other is an *iterative improvement method*, which improves a solution by means of local transformations.

The algorithm proposed here is composed of these two methods. The constructive one is called *SORG*, and the other one is called *GFDR*. The algorithm proceeds in the following way.

A Placement Algorithm

Step 1: Randomly generate a good initial solution by SORG.

Step 2: Improve the initial solution to obtain a locally optimum solution by GFDR.

Step 3: If available computation time has not been exhausted, go to *Step* 1; otherwise choose the best one among the locally optimum solutions obtained in *Step* 2.

This kind of hard combinatorial problem usually has many locally optimum solutions. One locally optimum solution might not be good enough. Therefore, repeating random generation of an initial solution and its improvement, we obtain a set of locally optimum solutions. The best one among them is chosen as a final solution.

A. SORG Method

SORG method selects modules, one at a time, based on an evaluation function which measures signal set connectivity to modules already or not yet selected, and then decides which slot the selected module will be placed on. Once a module is fixed into a position, it is not moved.

The conventional evaluation function, called IOC in [5], is adopted here. The module with the first or the second highest IOC is the logical candidate to be selected. Each of them is selected at *random* as the module to be put in place. The selected module is placed on the slot which yields the minimum total routing length among available slots. All available slots need not be examined here. Only a small part of them is examined, by calculating the ϵ-neighborhood of the median.

B. GFDR Method

Let S be the set of all feasible solutions and let x be a feasible solution, $x \in S$. Consider the neighborhood of x, denoted by $X(x)$, which is a subset of S. In the first step,

[1]This is not yet proved. However, the quadratic assignment problem, which is a particular case of the present problem, is proved to be NP-complete [12].

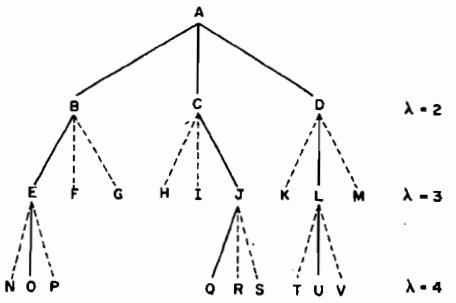


Fig. 1. Search tree.

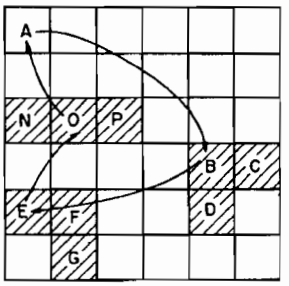


Fig. 2. Trial interchange of modules ($\lambda = 4$).

x is set to a feasible solution and a search is made in $X(x)$ for a better solution x' to replace x. This process, which is referred to hereafter as a *local transformation*, is repeated until no such x' can be found. A solution is said to be *locally* optimum if x is better than any other elements of $X(x)$.

A lot of definitions may be considered for the neighborhood of a solution. In [14], the set of solutions transformable from x by exchanging not more than λ elements is regarded as the neighborhood of x. A solution x is said to be λ-*optimum*, if x is better than any other solutions in the neighborhood in this sense.

Although the λ-optimum solution gets better as λ increases, the computation time easily goes beyond the acceptable limit, when an exhaustive search is performed for large λ. The following method does not examine all the elements in the neighborhood, nor does it guarantee a λ-optimum solution. However, it is very efficient in the sense that it can be applied for a large value of λ with limited searches in the neighborhood.

The present search procedure is illustrated along with the search tree shown in Fig. 1, where each node represents a module and each edge represents a trial transformation. The root node of the tree A is a module chosen to initiate the trial interchange, it is referred to as the *primary module*. A path connecting node A and one of the other nodes defines a possible interchange. For example, the path $A \rightarrow B \rightarrow E \rightarrow K$ refers to the trial interchange of four modules, as shown in Fig. 2. Here, module A is placed on the slot of B, B is placed on E, E on K, and K on A, in a round robin sequence. Although this transformation is a quadruple interchange, it includes a pairwise interchange as a special case, i.e., paths $A \rightarrow B$, $A \rightarrow C$, and $A \rightarrow D$, as shown in Fig. 3. Value λ indicates the number of modules to be interchanged.

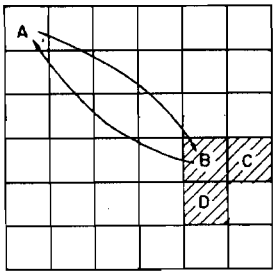


Fig. 3. Trial interchanges of modules ($\lambda=2$).

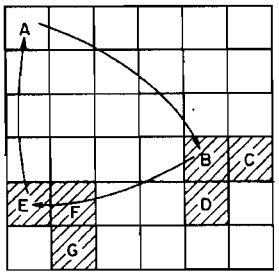


Fig. 4. Trial interchange of modules ($\lambda=3$).

The search tree is examined as follows. In this example, ϵ is fixed as 3. First, module A is interchanged with either one of the modules on trial in the ϵ-neighborhood of A median ($\lambda=2$). The ϵ-neighborhood modules are B, C, and D, thus pairwise interchanges between A and B, A and C, and A and D are performed. (See Fig. 3.) The trial interchange is accepted if it results in the reduction of the total routing length. If more than one reduction occurs in these transformations, the interchange with the greatest reduction is selected for acceptance. If no interchange contributes to reducing the total routing length, the next step ($\lambda=3$) is initiated.

Module A is placed on the slot of B. Then the median of B and its ϵ-neighborhood are calculated. In this case, the ϵ-neighborhood module are E, F, and G. Thus interchanges $A\rightarrow B\rightarrow E$, $A\rightarrow B\rightarrow F$, and $A\rightarrow B\rightarrow G$ are tried. (See Fig. 4.)

These trial interchanges are accepted if one of them results in the reduction of the total routing length. Otherwise, consider the three interchanges of paths $A\rightarrow B\rightarrow E$, $A\rightarrow B\rightarrow F$, and $A\rightarrow B\rightarrow G$, and choose the best one (least total routing length) for the later tree search. Here, $A\rightarrow B\rightarrow E$ is chosen, and $A\rightarrow B\rightarrow F$ and $A\rightarrow B\rightarrow G$ are omitted.

The solid lines in the tree search shown in Fig. 1 indicate which searches are to be continued. Broken lines show the searches which are to be terminated. Therefore, no more search efforts are made along paths $A\rightarrow B\rightarrow F$ and $A\rightarrow B\rightarrow G$. There is only one solid line under any node, except for root node A. Triple interchanges are performed for the other ϵ-neighborhood modules, C and D, of root node A. Tree search will be continued following J or L, whereas no search will be accomplished through H, I, K, and M. The tree search is continued, i.e., a path from node A is extended as long as λ is no greater than λ^*, which is given as a parameter.

TABLE I

	Example 1	Example 2	Example 3	Example 4	Example 5
#Modules	67	108	116	136	151
#Internal Modules	52	93	101	121	136
#External Modules	15	15	15	15	15
#Signal Sets	132	277	329	432	419
Av. #Signal Sets per Module	6.32	6.41	7.30	7.86	5.98
Av. #Modules per Signal Set	3.43	2.78	2.66	2.73	2.35
Range of Signal Set Size	2 - 9	2 - 9	2 - 9	2 - 9	2 - 9
Placement Grid Size	5 x 15	8 x 15	8 x 15	10 x 15	11 x 15

Each selection of a primary module is identified with an interchange *cycle*. Cycles are iterated until there is no reduction in the total routing length.

The GFDR method is different from the FDPR method in [7] which tries to interchange only a pair of modules. The GFDR, on the other hand, tries interchanges of more than two modules at the same time. If the interchanges are performed randomly in the GFDR, like in the PI method of [7], the gain will not compensate for the comparative great increase in computation time. The GFDR method examines and performs trial interchanges for subsets of modules which have a large possibility for improvement. This limited trial interchanges enable us to find a good locally optimum solution quickly.

Values ϵ and λ^* greatly affect the computation time and the total routing length. For a greater value of ϵ and λ^*, a better solution is obtained at the expense of computation time. In order to find a better solution within a given amount of computation time, the key problem is to determine how to set values ϵ and λ^*. In the following section, several experimental results are shown to point to the best values of ϵ and λ^*.

V. Experimental Results

In order to check and compare the results of the present algorithm with others, the algorithm was programmed and tested for five examples in a real problem. The program was written in FORTRAN and run on NEAC ACOS-77/700.

Example logic graphs were obtained from [13]. They represent five boards of the ILLIAC IV computer. The actual dimensions of ILLIAC IV boards are used as the grid size, i.e., there are 10×15 internal card locations and 15 I/O connector locations in one row on the bottom of the board. Grid size was reduced to the value shown in Table I. Statistics for the example logic graphs are also shown in Table I.

Experiment 1

The placement problem treated here has many locally optimum solutions. Thus different initial solutions lead to

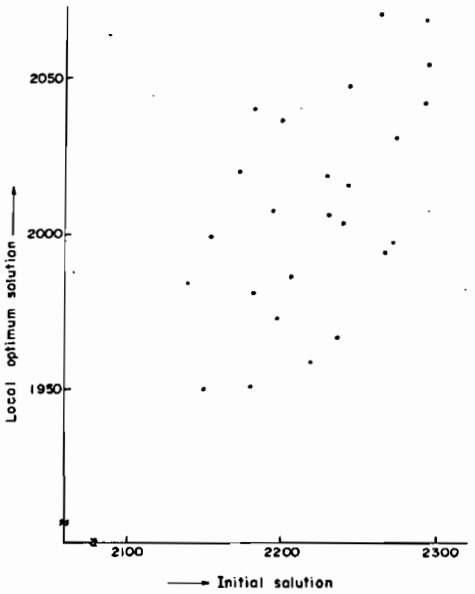


Fig. 5. Relation between initial solutions and locally optimum solutions.

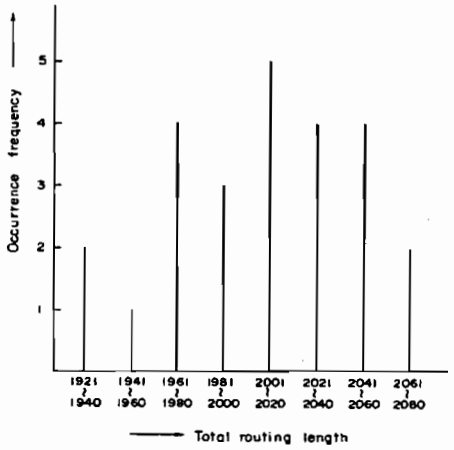


Fig. 6. Occurrence frequency for total routing length.

different solutions followed by an iterative improvement. It has been debated in the literatures whether it is better to use a random start or to use a constructive-initial solution, followed by an iterative-improvement. Experimental results in [6] and [7] showed that the constructive-initial start approach is superior to the random start in both solution value and computation time. Taking this fact into account, this constructive method generates *good solutions randomly* as initial solutions. In order to know the relation between the initial solution and its locally optimum solution, Example 5 was provided as a test with $\epsilon=4$ and $\lambda^*=4$, and 25 different initial solutions were generated by SORG and improved by GFDR.

Fig. 5 shows the relation between initial solutions and their locally optimum solutions. A better initial solution does not always result in a better locally optimum solution. This result does not recommend generating many good random starts and following *only the best one* with iterative improvement. But it recommends repeating random generation of an initial solution and its improvement to obtain *a set of locally optimum solutions*. The best one among them is chosen as a final solution.

Fig. 6 shows the occurrence frequency for the total routing lengths, which may be approximated by a normal distribution. The average value is 2007 and the standard deviation is 39.

Experiment 2

A simple transformation, such as pairwise interchange with smaller ϵ in general, does not yield better locally optimum solutions than more complicated ones with larger ϵ or λ^*. However, from the computational complexity point of view, the latter one takes much more time than the former one. In order to explore the influence of value ϵ or λ^* on computation time and the solution, the program was run by changing the value of ϵ or λ^*.

In Fig. 7, total routing length versus computation time curves are plotted for five values of ϵ operating on Example 5. Here, λ^* is fixed as 4. Each mark on the curves represents one cycle of GFDR method. The curve with smaller ϵ results in steeper descent, whereas it does not converge to a better solution. On the other hand, the curve with larger ϵ converges to a better solutions at the expense of computation time. A fairly small value of ϵ, i.e., $\epsilon=4$ or 5 is sufficient to lead to good solutions.

In Fig. 8, total routing length versus computation time curves are plotted for five values of λ^* operating also on Example 5, where $\epsilon=4$. The same discussion can be made for the value of λ^*, as for ϵ. The value $\lambda^*=3$ or 4 seems to be enough to have good solutions.

The aim of the overall scheme is to generate as many locally optimum solutions as possible within some time interval. Then, the best one among them is chosen. Let P be the probability that the cost of the locally optimum solution is less than V and let T_0 be the running time to produce the local optimum solution. Then, the best locally optimum solution produced within time interval T has a probability

$$\hat{P}=1-(1-P)^{T/T_0} \tag{6}$$

of being better than V [14], [15].

Random start were repeated as many times as possible within $T=30$ min for a value of ϵ. Here, Example 5 was provided as a test and λ^* was fixed as 4. Let $\hat{P}'_\epsilon$ be the probability of (6) associated with ϵ, and calculated $\hat{P}'_\epsilon$ of being less than 2000. The experimental results show the $\hat{P}'_2=0.68$, $\hat{P}'_3=0.71$, $\hat{P}'_4=0.95$, $\hat{P}'_5=0.92$, $\hat{P}'_8=0.70$. This procedure performs best at $\epsilon=4$.

The same analysis was done for finding the value of λ^*. Let $\hat{P}^*_\lambda$ be the probability of (6) associated with λ^* and calculated $\hat{P}^*_\lambda$ of being less than 2000. Here, ϵ was set to 4. We have the results that $\hat{P}^*_2=0.74$, $\hat{P}^*_3=0.93$, $\hat{P}^*_4=0.98$, $\hat{P}^*_5=0.89$, and $\hat{P}^*_8=0.75$. The best value was $\lambda^*=4$. The same tendency was observed for the other four examples.

Note 1: The algorithm with $\lambda^*=2$ did not result in good solutions in the meaning of both the appropriateness for solution value and computation time. This algorithm is called as the FDPR method in [7] and considered to be best among existing algorithms.

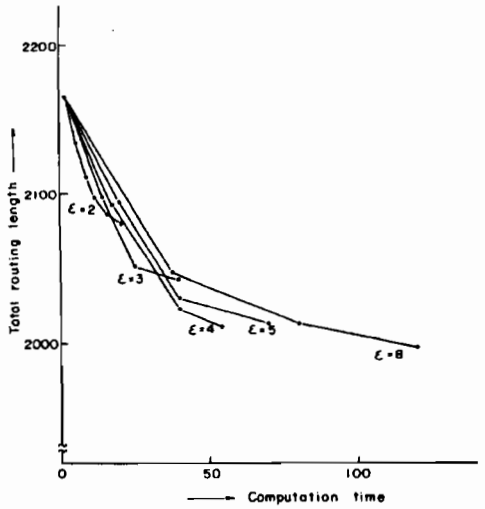


Fig. 7. Total routing length versus computation time curve for Example 5, varying the value of ϵ.

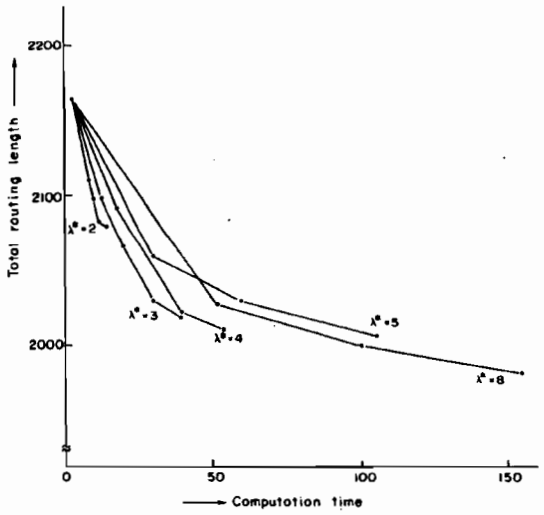


Fig. 8. Total routing length versus computation time curve for Example 5, varying the value of λ^*.

Note 2: The algorithm with larger ϵ did not result in good solutions either. The method with the testing of all possible interchanges is exactly equal to the proposed algorithm with the largest value of ϵ. In this sense, the testing of all possible interchanges is said to be inferior to limiting the number of possible exchanges proposed here.

Experiment 3

In order to explore the influence of the number of modules on the computation process, 5 examples were examined.

The graph in Fig. 9 shows computation time versus number of module for various values of ϵ. The computation time increases quite rapidly for $\epsilon > 6$, when the number of modules increases. Therefore, it seems infeasible to set $\epsilon > 6$ for large scale problems.

The graph in Fig. 10 also shows computation time versus number of modules, while varying the value of λ^*.

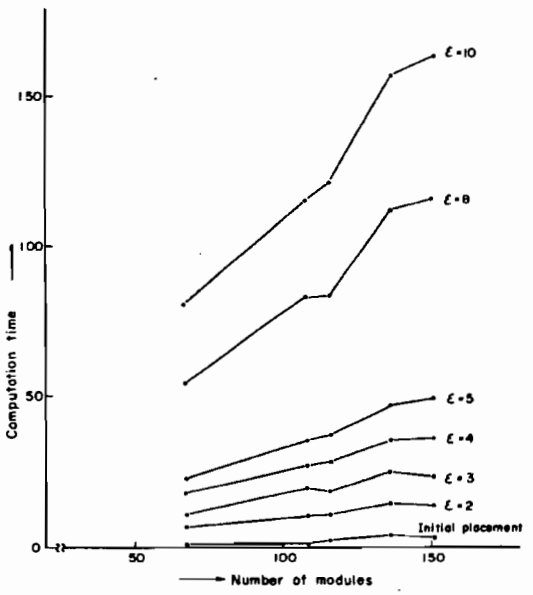


Fig. 9. Computation time versus number of modules, varying the value of ϵ.

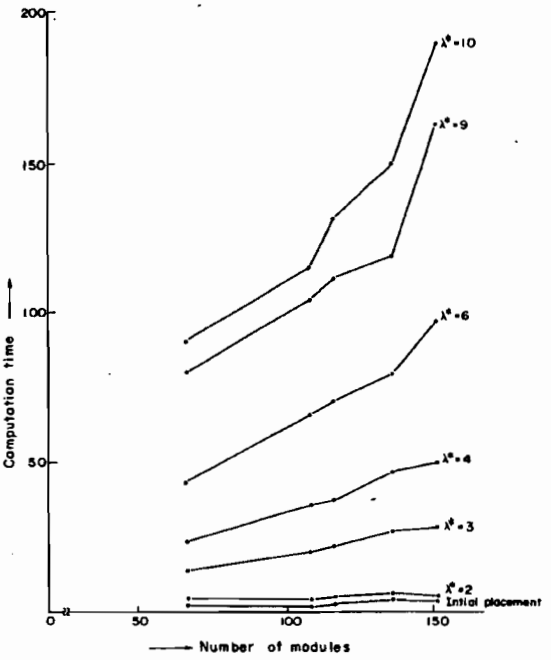


Fig. 10. Computation time versus number of modules, varying the value of λ^*.

From a practical point of view it may be reasonable to set $\lambda^* > 5$ for large scale problems.

The computation time to obtain a locally optimum solution with $\epsilon = 4$ and $\lambda^* = 4$ was linearly proportional to the number of modules.

VI. Concluding Remarks

In this paper, a new algorithm based on heuristic approach was proposed for the two-dimensional placement problem. The algorithm consists of two phases, namely:

initial constructive placement and iterative improvement. After repeating the random generation of an initial solution and its improvement by a sequence of local transformation, the best one among them is chosen as the final solution. The algorithm is greatly affected by the values of two parameters, ϵ and λ^*. From the present computational results, the algorithm performs best at $\epsilon=4$ and $\lambda^*=4$ in the meaning of both the appropriateness for solution value and computation time.

It should be noted that the algorithm with $\lambda^*=2$ did not result in good solutions, which is called as FDPR method in [7] and considered to be best among existing algorithms.

The most important point in the two-dimensional placement problem is to investigate various objective functions for optimum placement. Most of the present placement algorithms use routing length as an objective function, as was proposed in this paper. The "utilization of the most crowded channel", or "maximum density" is more meaningful for certain applications. One meaningful and perhaps more important criterion is to consider routability in placement.

Acknowledgment

The author is greatly indebted to Dr. E. S. Kuh, Dr. T. Ohtsuki, Dr. K. Kani, and H. Kawanishi for their helpful discussions and encouragements. Finally, he gratefully acknowledges the constructive comments of the reviewers who contributed to the improvement of an earlier version of this paper.

References

[1] L. Steinberg, "The background wiring problem: A placement algorithm," *SIAM Rev.*, vol. 3, no. 1, pp. 37–50, Jan. 1961.
[2] K. M. Hall, "An *r*-dimension quadratic placement algorithm," *Management Sci.*, vol. 17, no. 3, pp. 219–229, Nov. 1970.
[3] M. Hanan and J. M. Kurtzberg, "Placement techniques," in *Design Automation of Digital Systems; Theory and Techniques*. Vol. 1, (M. A. Breuer, ed.), Englewood Cliffs, NJ: Prentice-Hall, ch. 5, pp. 213–282, 1972.
[4] D. C. Schmidt and L. E. Druffel, "An iterative algorithm for placement and assignment of integrated circuits," in *Proc. 12th Design Automation Conf.*, pp. 361–368, 1975.
[5] D. G. Schweikert, "A two-dimensional placement algorithm for the layout of electrical circuits," in *Proc. 13th Design Automation Conf.*, (San Francisco, CA), pp. 409–414, 1976.
[6] S. Goto and E. S. Kuh, "An approach to the two-dimensional placement problem in circuit layout," *IEEE Trans. Circuits Syst.*, vol. CAS-25, pp. 208–214, Apr. 1978.
[7] M. Hanan, P. K. Wolff, and B. J. Anguli, "Some experimental result on placement techniques," in *Proc. 13th Design Automation Conf.*, (San Francisco, CA), pp. 214–224, 1976.
[8] S. Lin and B. Kernighan, "An effective algorithm for travelling-salesman problem," *Oper. Res.*, vol. 11, pp. 498–516, 1973.
[9] S. L. Hakimi, "Optimum locations of switching centers and the absolute centers and medians of a graph," *Oper. Res.*, vol. 12, pp. 450–459, 1964.
[10] D. B. Johnson and T. Mizoguchi, "Selecting the kth element in $X+Y$ and $X_1+X_2+\cdots+X_m$," *SIAM J. Computing*, vol. 7, no. 2, pp. 141–143, May 1978.
[11] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of Computer Computations*, (R. E. Miller and J. W. Thatcher, eds.), New York: Plenum, 1972.
[12] S. Sahni and T. Gonzales, "*P*-complete approximation problem," *J. ACM*, vol. 23, no. 3, pp. 555–565, July 1976.
[13] J. E. Stevens, "Fast heuristic techniques for placing and wiring printed circuit boards," Ph.D. dissertation Comp. Sci., Univ. of Illinois, 1972.
[14] S. Lin, "Heuristic programming as aid to network design," *Networks*, vol. 5, pp. 33–43, 1975.
[15] T. Yoshimura, "An algorithm for designing multi-drop teleprocessing networks," in *Proc. ICCC-78*, Oct. 1978.

Satoshi Goto (M'77) was born in Hiroshima, Japan, on January 3, 1945. He received the B.E. and M.E. degrees in electronics engineering from Waseda University, Tokyo, Japan, in 1968 and 1970, respectively, and the Ph.D. degree in engineering from Waseda University in 1977, for his research on computer-aided network design, graph theory, and combinatorial optimization method.

He joined Nippon Electric Company, in 1970 and is now a Supervisor of the Application System Research Laboratory, Central Research Laboratories. He has been engaged in the research and development of computer application system for LSI layout design, traffic control system and transmission or communication network design. He is now in charge of developing an advanced interactive CAD system for custom LSI and PWB. During 1975–1976 he was on leave at the Electronics Research Laboratories, University of California, Berkeley, where he worked on LSI layout and large-scale network design problems.

Dr. Goto is a member of the Institute of Electronics and Communication Engineers of Japan and the Operation Research Society of Japan.